Forest Explanation Through Pattern Discovery

Ca' Foscari University of Venice
Department of Environmental Sciences, Informatics and Statistics

Computer Science Master's Thesis
Academic Year 2019-2020

Graduand Alberto Veneri
Supervisor Prof. Lucchese Claudio

Abstract

In Machine Learning, some of the most accurate models are practically black-boxes, challenging to be interpreted and analyzed. Consequently, different strategies have been adopted to overcome these limitations, giving birth to a research area called Explainable Artificial Intelligence.
In this area, models considered black boxes are Deep Neural Networks, Support Vector Machines, and ensemble methods. In particular, ensemble methods based on trees are considered black-box models due to the multitude of trees being considered, even though they are singularly considered explainable.
Relevant techniques to explain ensemble of decision (regression) trees are now mostly based on methods that examine the features and outcome relationships, or create an explanation via tree prototyping, or locally approximate the model through explainable ones. Even though these approaches can give the end-user many meaningful insights into a model and its output, they do not produce a global model explanation by design and do not specify the type of interaction between features.
In this thesis, we move towards a new way of approaching the model explanation problem over an ensemble of regression trees by discovering frequent patterns inside the forest. A frequent patterns analysis produced from synthetic datasets created by basic algebraic functions has been performed to answer some initial questions: are there some frequent patterns related to a type of algebraic operation between features? If yes, what happens when the model tries to learn a function composed of basic operations? Multiple sub-problems have been addressed to answer the aforementioned issues and a framework has been proposed to solve them.
In our formulation, the relations discovered between the features are used to improve the accuracy of a tree ensemble adding to the training dataset new features that represent the association found. A quantitative evaluation has been made over real-world datasets comparing the accuracy of tree ensembles before and after applying the framework. The results show that the proposed technique can be a feasible solution to both improve and explain the model learned from a tree ensemble.

Keywords Explainable Machine Learning, Tree Ensemble, Pattern Discovery, Frequent Subtree Mining

Contents

Notations

$\mathbb{R}$	Set of real numbers
$P(\cdot)$	Probability Mass Function (PMF)
$p(\cdot)$	Probability Density Function (PDF)
$H(\cdot)$	Entropy of a r.v.
X	Random variable (r.v.), either discrete or continuos
$\mathcal{N}$	Normal distribution
$\mathcal{U}$	Uniform distribution
$\mathcal{X}$	Input space
$\mathcal{Y}$	Output space
$\mathbf{x}$	Sample in a multidimensional space
$\mathbf{x}_n$	N-th feature of a sample in a multidimensional space
y	Label of a sample $\mathbf{x}$ that belongs to $\mathcal{Y}$
$\hat{y}$	Estimate of a label
$\mathbb{1}$	Indicator function
T	Tree data structure
T'	Generic subtree of T

Introduction

Explainable Artificial Intelligence Over the last few years, Machine Learning (ML) has gained a lot of popularity in the computer science community to tackle problems that would be very difficult to be solved with handcrafted prediction models. Even though a lot of work has still to be done and there are a lot of open questions, ML is becoming more and more popular in many fields that affect our lives, such as health care, insurance policy, and autonomous driving among others. Due to its impact on our society, we are more and more interested in the understanding of the models generated by ML algorithms that in some cases are considered black-boxes. In fact, nowadays there is a need not only to perform well with some specific metric but also to understand what is going on under the hood to address some specific aspects, such as fairness and robustness against adversarial attacks. That is why, in the scientific literature and inside the commercial world, two terms in the ML field are becoming more and more popular, namely *interpretability* and *explainability*. Even though they are sometimes used as synonyms, they usually refer to two distinct characteristics of a model. The former is normally used to refer to an intrinsically interpretable model, where the latter is employed to indicate all the techniques that are used to make a black-box interpretable.

Explain tree ensembles In particular, in this thesis, we focus on the explainability of tree ensembles. Some techniques are already been developed to address some specific problems, such as the explanation of model outcomes, the analysis of the dependence between the features, or their importance in the model. However, after a thorough read of the scientific literature, it seems that a robust explanation of the model as a whole is missing. Specifically, we are interested to find a method that creates an arbitrarily complex function to describe the function learned by the ensemble. To tackle this problem, we argue if frequent patterns found in an ensemble could give us some insights between the type of feature interaction and if we can combine this information to explain the whole model.
By frequent patterns, we mean the frequent tree-based patterns that are present inside a forest, and the procedure of finding them is indicated as *frequent subtree mining*. Mining frequent subgraphs in an ensemble of graphs (we recall that trees are directed and acyclic graphs) is an NP-complete problem. However, during the years a variety of algorithms have been developed to mine frequent subtrees in ensembles of trees (also called dataset of trees) in an efficient way despite the computational explosion of the problem in large ensembles, or in ensembles composed of very deep trees.
Thus, in this thesis, we take advantage of two frequent subtree mining algorithms invented through the years, namely SLEUTH[57] and CMTreeMiner[55], to discover frequent patterns inside regression forests in order to find relations between

features and eventually explain the whole model using this information.

Besides, we investigate if the discovered relation between features can be used to improve the accuracy of the model, adding to the training set new features that represent such relations. In this way we try to address two goals: explain the model and, at the same time, improve its accuracy.

Proposed approach We structure our analysis with four research questions, which can be summarized as follows:

- Can we represent a forest with a simple closed-form expression?
- Can we create the closed-form expression (mentioned in the previous point) using the information retrieved from the frequent subtrees present in a forest?
- Can we discover the type of relationship between two features in a forest through frequent subtree mining?
- Can we improve the accuracy of a tree ensemble using the information retrieved from frequent pattern discovery?

To investigate possible solutions for our research questions, we create multiple datasets describing basic algebraic functions between paris of features, namely summation, product and division, and we analyze the frequent patterns present in the tree ensemble learned over these datasets.

From the preliminary analysis that we made in this thesis over synthetic datasets, we can say that the functions learned by a tree ensemble are strongly linked with the frequent subtrees present in the model. This means that, if a tree ensemble is modeling a simple algebraic function, we can classify it through its frequent subtrees.

We have then investigated the possibility to use the information of the presence (or absence) of frequent subtrees in a particular function to find the same relation between features in more complex function. In this way, we wanted to understand if the patterns that are generated are in some way composed by simpler patterns. In this case we found that there are some indications suggesting that we could be able to classify the relation between features given frequent subtrees mined from an ensemble, even though further investigation are needed to identify if a real correlation exists.

Furthermore, assuming that the relations between features found in complex function are generally correct, we analyzed if it possible to improve the accuracy of the ensemble adding a feature to the dataset representing the new relation discovered and then re-train the model. In this case we found that in most of the case we are able to improve the accuracy of a model trained dover specific synthetic datasets.

Finally, we evaluate the approach proposed over four real world dataset, finding that in most of the case we are able to improve the accuracy of a model adding new features that represent the relations discovered through frequent subtree mining.

Outline of the thesis Specifically, the thesis is organized into six main chapters, starting from describing the background necessary to understand the proposed framework and ending with the conclusion and future works. In chapter 1, we present some basic concepts related to ML, focusing on describing the model of interest of this thesis, i.e. tree ensembles. Then, in chapter 2 we bring our attention to the Explainable Artificial Intelligence field, which is at the core of our investigation, describing the motivation behind it and the terminology used. In chapter 3, we present the state-of-the-art explanations techniques used with tree ensembles.

After the first introductory part, in chapter 4 we present an analysis of frequent subtrees in tree ensembles that are used to explain the function learned by the model. In addition, we propose a framework to mine frequent subtrees from a tree ensemble and classify the relationships between the features. Then, in chapter 5 we illustrate an evaluation of the proposed framework over four real-world datasets, and we show that our framework can improve the accuracy of the models and we also present a case study to show that the relations found can be also meaningful. Finally, in chapter 6 we discuss the results obtained, describing the new findings, discussing limitations and drawbacks, and present possible future investigation over this topic.

Chapter 1

Machine Learning Background

The purpose of this chapter is to recall some essential techniques, definitions, and algorithms that are well known in the ML field and are strictly linked to the topics in this thesis. Starting from some basic concepts from *information theory*, we introduce the terminology used to describe a *supervised machine learning problem*, and we end up with an introduction to *ensemble methods*.

1.1 Information Theory Aside

This section briefly summarizes some significant results taken from the information theory field, which are at the core of the techniques discussed in the following sections. We are mainly interested to recall the definitions of *information*, *entropy*, and *mutual information*.
Given a discrete random variable (r.v.) X associated to an alphabet $\mathcal{A}_{\mathrm{X}} = \{a_1, ..., a_I\}$ that represents all the possible outcomes, having probabilities $\mathcal{P}_{\mathrm{X}} = \{p_1, ..., p_I\}$, such that $P(x = a_i) = p_i$, in information theory, the definition of information of an outcome a_i is strictly related with its probability $P(x = a_i)$, as defined below[36].

Definition 1 (Information of an outcome)**.** The information of an outcome x is equal to the inverse of the logarithm of its probability.

$$h(x) \coloneqq \log_2 \left(\frac{1}{p(x)} \right)$$

Therefore, if $P(x) \approx 0$ the associated information will be high, and, on the other hand, if $P(x) = 1$ means that $I(x) = 0$. It is also relevant to notice that the logarithm base represents the unit of the information, which in general is equal to 2 (a bit).
Strictly related to the notion of information, there is the notion of entropy.

Definition 2 (Entropy of a random variabile)**.** The entropy of a discrete r.v. X is defined as the weighted average of the information of all possible outcomes of the variable:

$$H(X) \coloneqq \sum_{x \in \mathcal{A}_{\mathrm{X}}} p(x)h(x) = -\sum_{x \in \mathcal{A}_{\mathrm{X}}} p(x) \log_2 p(x)$$

In the case when $p(x) = 0$, the convention is that $p(x)h(x) = 0 \log 0 = 0$, justified by the fact that $\lim_{x \to 0} x \log x = 0$.
Another fundamental quantity in information theory is the *relative entropy* or *Kullback-Leibler divergence* $D_{\mathrm{KL}}(P \| Q)$.

Definition 3 (Kullback-Leibler divergence)**.** The Kullback-Leibler divergence between two probability distribution $P(x)$ and $Q(x)$ defined over the same alphabet $\mathcal{A}_{\mathrm{X}}$.

$$D_{\mathrm{KL}}(P\|Q) \coloneqq \sum_{x\in\mathcal{A}_{\mathrm{X}}} P(x)\log\frac{P(x)}{Q(x)}$$

Directly derived from the relative entropy, it is defined the *mutual information* $I(\mathcal{X};\mathcal{Y})$, between two discrete r.v. X and Y.

Definition 4 (Mutual Information or Information Gain)**.** The mutual information between two discrete r.v. X and Y, is defined as the Kullback-Leibler divergence between the joint probability distribution $P_{(\mathrm{X},\mathrm{Y})}$ and the product of the probability distributions $P_{\mathrm{X}} \otimes P_{\mathrm{Y}}$.

$$\begin{aligned} I(\mathrm{X};\mathrm{Y}) &\coloneqq D_{\mathrm{KL}}\left(P_{(\mathrm{X},\mathrm{Y})}\|P_{\mathrm{X}}\otimes P_{\mathrm{Y}}\right) \\ &= \sum_{y\in\mathcal{Y}}\sum_{x\in\mathcal{X}} p_{(X,Y)}(x,y)\log\left(\frac{p_{(X,Y)}(x,y)}{p_X(x)p_Y(y)}\right) \end{aligned}$$

The mutual information has some interesting properties linked with the entropy defined as above. In fact it is easy to show that:

$$\begin{aligned} I(\mathrm{X};\mathrm{Y}) &= H(\mathrm{X}) - H(\mathrm{X}|\mathrm{Y}) && (1.1) \\ &= H(\mathrm{X}) - H(\mathrm{Y}|\mathrm{X}) && (1.2) \end{aligned}$$

The above equation means that the mutual information actually computes the amount of information we gain when we know one of the two random variables. Consequently, $I(\mathrm{X};\mathrm{Y}) = 0$ if and only if X and Y are independent.
Even though all the above results are defined on discrete random variables, it is easy to generalize all the results to the continuous case.

1.2 Supervised Learning Problem

The supervised learning problem can be described informally as the problem of finding the unknown relationship between a set of features that describe an object and its label starting from some observations. A typical example is a handwritten digit classification problem: given some samples of grayscale images with resolution 128×128, represented in a space $\mathcal{X} = [0,1]^{128\times128}$, and their associated labels, represented in a space $\mathcal{Y} = \{0,1,...,9\}$, the objective of a supervised machine learning problem is to find a function f that better maps $\mathcal{X}$ into $\mathcal{Y}$.
More formally, in a supervised machine learning problem we characterize two spaces: the feature space $\mathcal{X}$ (also called the input space) and a label space $\mathcal{Y}$ (also called the output space).

Classification and regression In the feature space $\mathcal{X}$ each dimension represents a particular feature, each of which can be either discrete or continuous. Moreover, these represent the features that describe a certain point (or sample) in the feature space. Similarly, the output space $\mathcal{Y}$ can be either discrete or continuous. $\mathcal{Y}$ is discrete if our supervised learning problem is a *classification* problem, e.g. if we have a feature space that represents some animals' characteristics such as weight, length, and the presence of mustache or not, and we want to know if

a point in X is a cat or not. Specifically, this is a binary classification problem, and the output space it can be represented as $\mathcal{Y}_b = \{1, 0\}$ where 1 means that the animal is a cat, 0 the opposite.
On the other hand, the problem is called a *regression* problem when $\mathcal{Y}$ is a continuous space. For example, a problem in this category could be to predict the selling price of an house given its position, its size, the number of house in the neighborhood and so on.

The learning problem Given $\mathcal{X}$ and $\mathcal{Y}$, we want to find a function f that represents the unknown relationship between these two spaces. Since we do not know $f(x)$ in advance, we just estimate a $\hat{f}(x)$ that is a good approximation of $f(x)$. In particular, in the supervised learning setting, $\hat{f}(x)$ is built on the basis of a training set D_N, $\hat{f}(x) = g(x, D_N)$. D_N is formed by N samples $D_N = \{(x_i, y_i)\}_{i=1}^{N}$ with $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. When a sample is described with more than a single feature, as in the following subsections, the notation $\mathbf{x} \in \mathcal{X}$ is used.
In addition, given a true label $y \in \mathcal{Y}$ and a label $\hat{y} \in \mathcal{Y}$ predicted by the estimated function $\hat{f}(x)$, we define a loss function $l : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ as a measure of accuracy of our estimates. Popular choice of loss function are the 0/1 loss function for classification problem, $l_{0/1}(\hat{y}, y) = \mathbb{1}\{\hat{y} \neq y\}$, where $\mathbb{1}$ is the indicator function, and the squared error loss function for the regression $l_2(\hat{y}, y) = (\hat{y} - y)^2$.
In this thesis we mainly considered regression problems and we use the squared error loss as standard loss function.

The learning phases It is common to divide the learning phase into two parts, namely training, and testing. During the *training phase* the function $\hat{f}(x)$ is learned from a *training set* D_{tr}, with $D_{tr} \subset D$, following the learning procedure of choice. At the end of this phase, to verify the quality of the approximation, the remaining part of the dataset D_{te}, or *test set*, defined such that $D_{te} \cup D_t = \emptyset$, is used to verify the accuracy of $\hat{f}(x)$. Thus, during the *test phase*, for each sample $\mathbf{x} \in D_v$ the predicted label $\hat{y}$ and the value of the loss function for that prediction $l_2(\hat{y}, y)$ are computed. Finally, a weighted sum of the loss for each prediction is calculated to represent the accuracy of the model created.
This division between training and test set is useful to catch the *generalization error* of the model, i.e. it is a measure to verify how accurate are the predictions of the estimated function with new and unknown samples. If a model performs well only on the training set and not on the test set, it means that it generalizes poorly and this problem is called *overfitting*.
Before the testing phase, it is usually also needed to tune some parameters that cannot be learned during the learning phase, which are called *hyperparameters* and are used to control the learning phase. The tuning of the hyperparameters is usually called *validation phase* and it is normally done in another partition of the data different from the training and test set called *validation set*, indicated in this thesis with D_v.

1.3 Classification and Regression Trees

Classification and regression trees are standard ML models adopted to tackle a supervised learning problem that are either reasonably simply to implement and intelligible. In these models, at the end of the training phase a decision tree is

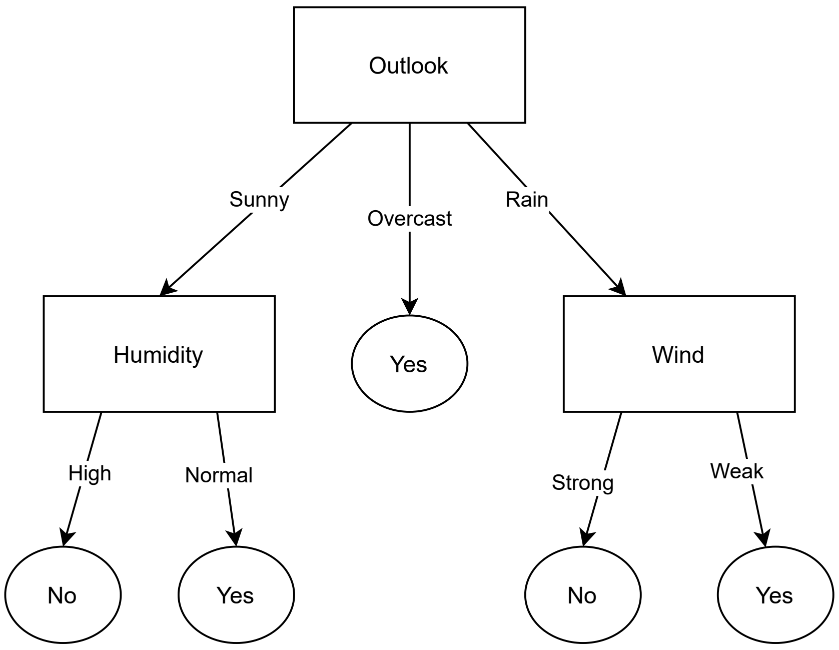


Figure 1.1: Classical "play tennis" decision tree.

formed and it is used to predict a label (in case of classifications trees) or a value (in case of regression trees) for a new sample. In the following subsection, an overview and the terminology belonging to classification and regression trees and some well-known learning algorithms are presented.

1.3.1 Overview and notation

The decision tree created after the learning phase is based on a tree data structure. A tree data structure is an acyclic, connected, and directed graph $G = (V, E)$ where V is the set of vertices $V = \{v_i\}_{i=1}^{N}$, and E is the set of edges $E = \{e_i\}_{i=1}^{M}$. Furthermore in every tree, we can define for each edge e_i two functions: $parent(e_i) \in V$ and $children(e_i) \in V$, representing the parent-child relationship between nodes. In a tree, every child node is a child of a single parent but a parent node can have multiple children. Furthermore, a node is called leaf if it has no children, or inner node otherwise.

In a decision tree, each non-leaf node is associated with a predicate, also called a *split*, that it is used to separated the data according to the result of the test. For example, in Figure 1.1, there is a representation of a possible decision tree learned to predict if a person would like to play tennis give information about the weather outlook, the air humidity, and wind. In this case the leaves of the decision tree are represented with ellipses and the inner nodes with rectangles. To predict if a player is going to play tennis with under a certain setting, it is jus necessary to follow the corresponding arrows at each split and reach a final leaf. For example if their is a sunny day with a normal level of humidity, this classifier predicts that a tennis player would like to play tennis.

1.3.2 Learning Algorithms

During the years, a lot of different techniques have been proposed to improve the accuracy of regression and decision trees learned from data. Even tough various techniques are available to select the splitting criteria and to prune the trees, the main structure of the learning algorithm can be generalized in the following three steps:

1. Take as input a dataset D and create the root of the tree
2. For each feature, find a split of the dataset D that minimize a specific impurity measure. Then, chose a split with the minimum impurity associated overall and create two datasets D_1 and D_2, they will be the children of the root of the tree.
3. If the stopping criteria have not been reached, repeat from step 2 for each child node in turn.

In the description above, the impurity measure plays a key role in the determination of the final tree. In particular, the major algorithms for decision and regression tree learning mainly differ for the type of impurity measure (also called splitting criterion) they use.
As also depicted from the article written by Loh[33], among the most popular learning algorithms to build classification trees there are ID3[40], C4.5[41], and CART[8]. All the algorithms have the same structure that follows the one described at the beginning of the section: the dataset is recursively partitioned following specific criteria, until particular stopping criteria have been reached. The main difference between the three algorithms is the choice of the specific criterion used for the split, in particular the information gain is adopted in ID3, the gain ratio (a variant of the information gain) in C4.5, and the Gini index in CART. In these three methods for classification, the prediction of a new sample $\mathbf{x}$ is decided following the decision path accordingly to the values of the features of $\mathbf{x}$ until reaching a leaf. Then, the class prediction of $\mathbf{x}$ is assigned based on the most frequent class in the portion of the dataset associated with that leaf.
Regarding the algorithms designed for learning regression trees, we highlight CART for regression trees, and M5[42]. The CART algorithm used for a regression problem is similarly defined as the one for the classification problem. The differences from the implementation for the classification problem are two: the computation of the impurity of each node and the way of making a prediction from a leaf. The impurity is normally computed as a sum of squared deviations from the mean, thus the impurity $i(v_i)$, where v_i is the current node and to v_i is associated the subset D_i of the dataset, is calculated as $i(v) = \sum_{\mathbf{x} \in D_i} (\mathbf{x} - \bar{\mathbf{x}})^2$, with $\bar{\mathbf{x}} = \frac{1}{N_i} \sum_{\mathbf{x} \in D_i} \mathbf{x}$ and N_i the cardinality of the set D_i. CART is a piecewise-constant model because the value predicted from a leaf is constant and it corresponds to the average of the labels in that split. M5 is instead a learning algorithm developed by Quinlan that creates a piecewise-constant linear model generating, first of all, a constant tree and then fits a linear regression model to the data in each leaf.

1.4 Ensemble Methods

With *ensemble methods* we normally refer to a set of techniques used to construct a set of N weak learners and then combine them to form a prediction model. A simple diagram with the general framework is shown in Figure 1.2.
This learning method is sometimes also called *committee-based learning*, or *learning multiple classifier*[59].

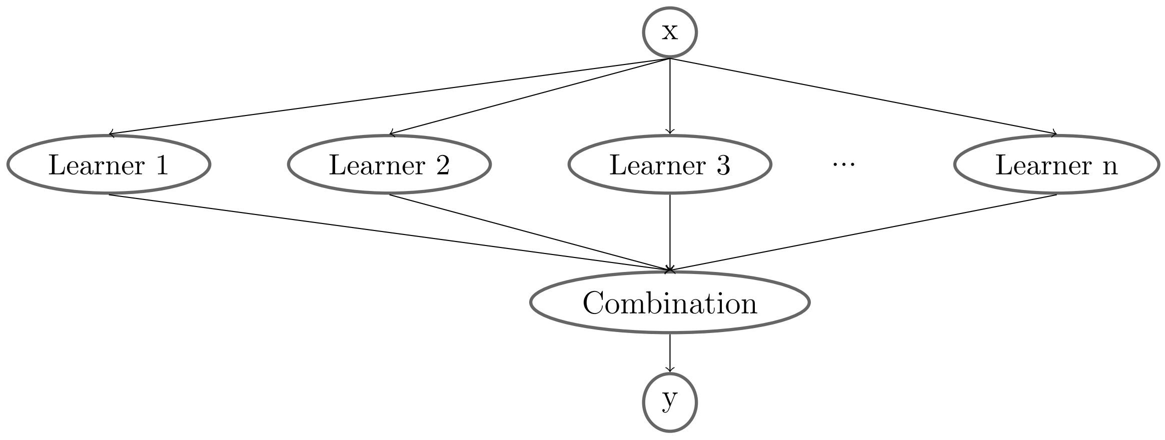


Figure 1.2: Ensemble methods, general framework. From a feature vector $\mathbf{x}$ we train N classifiers and then their models are combined to get the output y

Two approaches for ensemble learning are presented in this section: *boosting* and *bagging*.

1.4.1 Boosting

With the term *boosting* procedure we refer to an ensemble learning method that boosts the accuracy of a weak learner, that by definition it has an accuracy that is only slightly better than a random guess, into a strong learner.
The equivalence between the weak learnability model and the strong learnability model was proved in 1990 by Schapire [47], and it has been the first of a series of results for the wide-spreading of the *boosting* approach. The first boosting algorithm (AdaBoost) has been introduced by Freund and Schapire in 1997[18] with the aim of transforming a weak classifier obtained with a Probably Approximate Correct (PAC) learning algorithm into a strong classifier with arbitrarily high accuracy. AdaBoost can be seen as one of the most influential boosting algorithm developed, and it has been the precursor of all the other boosting procedures.
The general boosting procedure is sequential and divided in T_{boost} rounds, where at each round a new weak learner $\hat{f}_t$ is introduced aiming to correct the mistakes committed by the ones generated in the previous round. In order to allow the new learner to correct the mistakes of the previous, a new distribution D_t of the data is generated. To achieve this improvement, in D_t the "weight" of the samples misclassified are increased. At the end of these T_{boost} rounds, the output is provided with a combination of each weak learners' outputs.
In Algorithm 1, a pseudocode for the the general boosting procedure is presented [59].

1.4.2 Bagging

The algorithm, originally introduced by Breiman in 1996[6], runs in T_{bagg} turns, and at each turn a random sample with replacement of size m is created from the initial training set D. With every sample, a new learner is trained, and, at prediction time, all the learners' vote is aggregated. If, for example, the outcome y is numerical, the aggregation could simply be an average of each vote.

Algorithm 1 General Boosting Procedure

```
1: procedure Boosting($D, T, L$)
2:     ▷ Dataset $D$, learning algorithm $L$ and $T$ rounds
3:     $D_1 \leftarrow D$
4:     for $t = 1, ..., T$ do
5:         $\hat{f}_t \leftarrow L(D_t)$
6:         $\epsilon_t \leftarrow evaluate\_error(h_t(x), f(x))$
7:         $D_{t+1} \leftarrow create\_new\_dist(D_t, \epsilon_t)$
8:     end for
9:     return $combine\_learners(h_1(x), ..., h_t(x))$
10: end procedure
```

The whole procedure, since it is based on random sampling with replacement of the initial dataset, also called bootstrapping in statistics, has been called "**b**ootstrap **agg**regat**ing**", or simply with its acronym: bagging.
A general framework for a bagging procedure is presented in Algorithm 2.

Algorithm 2 General Bagging Procedure

```
1: procedure Bagging($D, T, L$)
2:     ▷ Dataset $D$, learning algorithm $L$ and $T$ rounds
3:     for $t = 1, ..., T$ do
4:         Draw a sample $D_s$ from $D$
5:         $\hat{f}_t \leftarrow L(D_s)$
6:     end for
7:     return $combine\_learners(h_1(x), ..., h_t(x))$
8: end procedure
```

1.5 Popular Ensemble Methods

Among all the methods developed since the introduction of the boosting and bagging frameworks, a few of them are quite popular and achieve excellent results in many competitions[46], specifically they are AdaBoost, Bagging, Random Forest, and Gradient Boosting Machines. In this category of well known and used methods, we focus only on two of them, namely Gradient Boosting Machines (GBM), and Random Forests (RF). In this section, we briefly revise the main characteristics of these two methods, with an emphasis on the learning algorithm and the type of layout of the forests generated.

1.5.1 Gradient Boosting Machines

Gradient Boosting Machines were introduced by Friedman in 1999[19], and are models derived from boosting learning algorithms that try to transform the boosting learning problem into an optimization one. As informally showed in subsection 1.4.1, the final function produced by a boosting procedure can be seen as a combination of functions of weak learners. Taking a generic loss function $l(y, \hat{y})$, where $\hat{y} = \hat{f}(x)$, the boosting procedure produces a function

$$\hat{f}(\mathbf{x}) = \sum_{t=1}^{T} \beta_m h(\mathbf{x}; \mathbf{a}_t) \tag{1.3}$$

where, in this case, $\{\beta_t\}_1^T$ are the weights of each learner and $\{\mathbf{a}_t\}_1^T$ are the parameters of each base learners. Thus, the combination of the base learners $h(\mathbf{x}; \mathbf{a}_t)$ at the end of the algorithm, in this case, is simply a weighted sum.
The boosting algorithm can be seen as a stage-wise learning, where at each step t a new prediction function is formed, specifically $\hat{f}_t(\mathbf{x}) = \hat{f}_{t-1}(\mathbf{x}) + \beta_t h(\mathbf{x}; \mathbf{a}_t)$, at each stage we can formalize an optimization problem of the type

$$(\beta_t, \mathbf{a}_t) = \arg\min_{\beta, \mathbf{a}} \sum_{t=1}^{T} l(\mathbf{y}, \hat{f}_{t-1}(\mathbf{x}_i) + \beta h(\mathbf{x}, \mathbf{a}_t)) \tag{1.4}$$

That is solved in two stages, where the first one solve a least squares optimization problem to find $\mathbf{a}_t$ and the second one is a simple one parameter optimization problem used to find the weight β_t. Formally the former optimization problem is the following one

$$\mathbf{a}_t = \arg\min_{\mathbf{a}, \rho} \sum_{i=1}^{N} [\tilde{y}_{it} - \rho h(\mathbf{x}_i; \mathbf{a})] \tag{1.5}$$

where $\tilde{y}_{im}$ is the so-called pseudo-residual

$$\tilde{y}_{it} = \left[\frac{\partial l(y_i, \hat{f}_{t-1}(\mathbf{x}_i))}{\partial \hat{f}_{t-1}(\mathbf{x}_i)} \right] \tag{1.6}$$

and ρ is the learning rate.
Thus, after this step we have found all the parameters $\mathbf{a}_t$ for the weak learner $h(\mathbf{x}, \mathbf{a}_t)$, therefore we are able to formalize also the second optimization problem to find its weight.

$$\beta_t = \arg\min_{\beta} \sum_{i=1}^{N} l(y_i, \hat{f}_{t-1}(\mathbf{x}) + \beta h(x_i, \mathbf{a_t})) \tag{1.7}$$

The procedure described above is general and suitable for every type of weak learners and loss functions, in Algorithm 3 we present instead the GBM algorithm in the case where the base learner is a regression tree, namely Gradient Boosting for Regression Trees (GBRT) as originally proposed by Friedman[20]. In GBRT, since each weak learners is a regression tree, at each step t and in the first stage, L regions $\{R_{lm}\}_1^L$ are defined corresponding to the L leaves of the decision tree created during the learning procedure based on the dataset produced each time on the pseudo-residuals $\{\tilde{y}_{it}, \mathbf{x}_i\}_{i=1}^N$. In the second stage, assuming that the final model is a constant piecewise model, as described in subsection 1.4.1 for the CART algorithm, to minimize Equation 1.7, we just have to find a value γ_{lt} for each region that added to the previous estimated function $\hat{f}_{t-1}(\mathbf{x})$ minimize the equation. Therefore, the new problem becomes

$$\gamma_{lt} = \arg\min_{\gamma} \sum_{\mathbf{x}_i \in R_{lt}} l(y_i, \hat{f}_{t-1}(\mathbf{x}_i) + \gamma) \tag{1.8}$$

The description of the algorithm aforementioned follow the explanation proposed by the same Friedman in a variant of his initially proposed algorithm[20].

Algorithm 3 Gradient Boosting Regression Trees

```
1: procedure GBRT($D, T$)                                   ▷ Dataset $D$, $T$ rounds
2:     $\hat{f}_0(\mathbf{x}) \leftarrow \arg\min_\gamma \sum_{i=1}^{N} l(y_i, \gamma)$
3:     for $t = 1, ..., T$ do
4:         $\tilde{y}_{it} \leftarrow -\left[\frac{\partial l(y_i, \hat{f}_{t-1}(\mathbf{x}_i))}{\partial \hat{f}_{t-1}(\mathbf{x}_i)}\right]$
5:         Learn the tree $\{R_{lt}\}_1^L$
6:         $\gamma_{lt} \leftarrow \arg\min_\gamma \sum_{\mathbf{x}_i \in R_{lt}} l(y_i, \hat{f}_{t-1}(\mathbf{x}_i) + \gamma)$
7:         $\hat{f}_t(\mathbf{x}) \leftarrow \hat{f}_{t-1}(\mathbf{x}) + v \cdot \gamma_{lm} \mathbb{1}(\mathbf{x} \in R_{lm})$
8:     end for
9:     return $\hat{f}_T$
10: end procedure
```

1.5.2 Random Forests

An improved version of bagging are the Random Forests (RF). The main idea of Random Forests' technique has been introduced independently at about the same time by Ho[51] and Amit and Geman[3], and then popularized by Brieman in 2001[7]. In his influential paper, Brieman proves that a random selection of the features during the training phase of a weak learner can improve the generalization and the accuracy of bagging. Since that the number of feature selection dramatically effect the accuracy of each classifier, it has also been suggested to do a internal estimates of the generalization error using the so-called out-of-bag estimates, i.e. evaluate the weak model against the samples not included during the bootstrap procedure, to find the most accurate randomization. The algorithm is basically the one described in Algorithm 2, but instead of using the given implementation of a learning algorithm L, it uses a modified version of L where at each split a random sampled subset of the features is used. For example, one of the implementations proposed by Breiman called *Forest-RI* uses CART as algorithm L, and at each split, it uses a subsample of $\lfloor \log_2 M + 1 \rfloor$ features, where M is the initial number of features.

1.6 Conclusion and summary

Conclusion In this chapter, we have briefly reviewed some basic results from information theory and machine learning that are necessary to follow the investigation over a possible new method that we propose to explain a tree ensemble. In particular, in the initial part, we have described in detail the computation and the interpretation of the information gain because it has been used to investigate the discriminative power of the features in some parts of our proposed method. We have also described in-depth the algorithmic details of GBM and RF because, in addition to being very popular in the machine learning community, they are also the two types of tree ensemble that we have studied. This introductory part must be seen as a preamble and a brief recap for the following analysis where the main goal is to try to explain the two type of tree ensembles presented in this chapter, that are in general considered as black-boxes.

Summary The summary of the chapter divided by sections is the following:

- In section 1.1 we have illustrated four main definitions described in the information theory, namely the information of an outcome (Definition 1), the entropy of a random variable (Definition 2), the Kullback-Leibler divergence (Definition 3), and the mutual information (Definition 4).
- In section 1.2 we have presented one of the most known problem in the machine learning field, the supervised learning problem, where an algorithm is implemented to discover the function that maps the input space $\mathcal{X}$ to the label space $\mathcal{Y}$. We have reviewed the difference between a classification and regression problem and the different phases of the learning process.
- In section 1.3 we have presented the definitions of classification and regression trees and we summarized the main characteristics of four learning algorithms, namely ID3, C4.5, CART, and M5.
- In section 1.4 we have described the structure of and the general idea behind an ensemble method. We have also illustrated two main frameworks for its implementation: boosting and bagging.
- In section 1.5 we have presented two actual implementations for bagging and boosting, namely Gradient Boosting Machine (GBM) and Random Forest (RF).

Chapter 2

Explainable Artificial Intelligence

Artificial Intelligence (AI) and Machine Learning (ML) are becoming prominent in all the sectors of our economy, and in our everyday life[45]. Even though a lot of research on these fields has been carried out during the second half of the 20th century, only during the last two decades there has been a widespread use of AI-powered technologies outside the research labs. One of the reasons for this success can be found in the increasing accuracy of the more recent models that was made possible thanks to the explosion of computational power, the development of general-purpose programming on graphics processing units, and the availability of an enormous amount of data. However, one of the main downsides of the last and more used machine learning technologies, such as *Deep Neural Network* (DNN) and ensemble methods, is that they are very complex models to be understood by humans. They are not so interpretable because they adopt a huge number of parameters. Due to these characteristics, they are considered *black-box models.*

Hence, with the increment of their usage, it is rising the request from all the different stakeholders for more "transparent" and trustworthy AI algorithms and technologies. This is not only an emerging trend, but is also determined by law in some jurisdictions. For example, in all the European countries since the approval of the General Data Protection Regulation (GDPR), written by the European Union Parliament and Council in 2016, every citizen could avail their "Right of explanation". This means that users can ask for an explanation of an algorithmic decision that significantly affects them[22]. Examples of fields where this right can be applied are autonomous driving, medical diagnosis, and insurance risk assessment among others.

For all the above reasons, a new interdisciplinary field has emerged during le last few years (see Figure 2.1) getting a lot of attention: *Explainable Artificial Intelligence* (XAI). It is still an emerging field and there are a lot of open questions[1], thus in this chapter we try to summarize all the main points of XAI. We start with a brief introduction to the field focusing on the rationales behind XAI and the terminology used. Finally, we present the main types of techniques that have been proposed during the years to explain ensemble methods and how to measure the quality of the explanation developed.

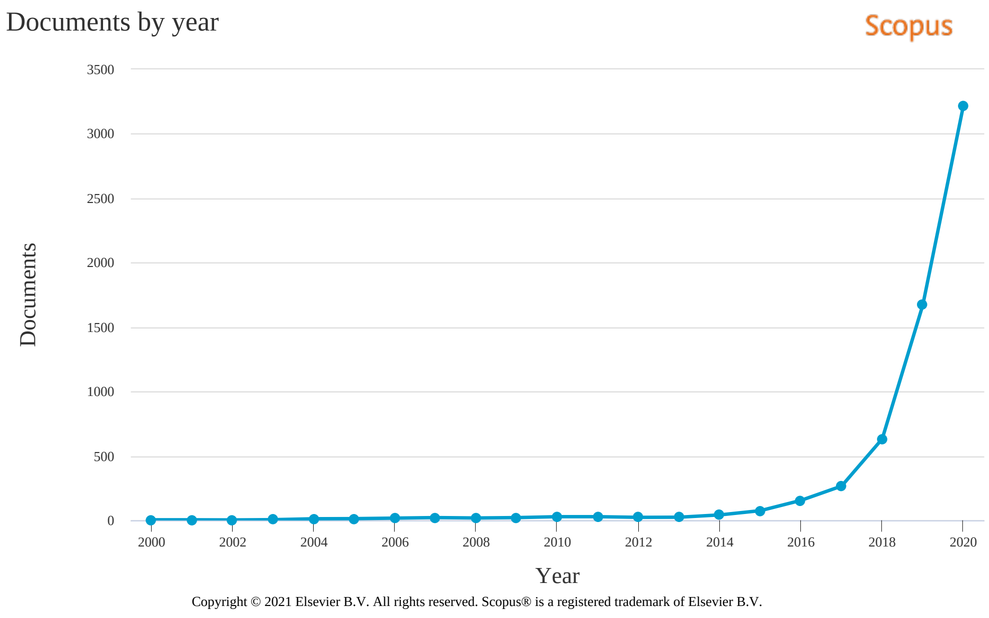


Figure 2.1: Chart showing the explosion of interest in the XAI field during the years through an analysis of the results from Scopus over the peer-reviewed paper published. The query used was *explainable AND artificial AND intelligence*, over all the possible fields, and within the years 2000-2020.

2.1 Definitions

Across the literature, various definitions of common terms, such as *explainable* and *interpretable*, have been used with the respect to the XAI field. Depending on the field of application, also the term Explainable Artificial Intelligence has been used interchangeably with other terms, that, at first glance, seem synonymous, e.g. Interpretable Artificial Intelligence. To avoid any confusion between the terminology used, we present in Table 2.1 all the relevant definitions used across the literature, merging information provided by the major surveys available in the field [23][1][5]. In particular, we characterize the difference between black-box models and transparent models, and among Interpretable AI, Explainable AI, and Responsible AI.

2.2 XAI a Multidisciplinary Field

Explainable Artificial Intelligence is a topic of interest for four main communities as suggested from the preprint of Mohseni et al.[37]: social science, Human-Computer Interaction (HCI), visual analytics, and machine learning. In this thesis we combine together HCI with visual analytics, therefore creating three main areas, as pictorially presented in Figure 2.2. Each community has adopted different design goals and have been essential to the development of XAI. In this section we review, for each field, the typical designs guidelines used with the aim to understand how the design and evaluations are typically done with respect to different communities, highlighting also the necessity for each new type of technology developed in the XAI field to address different problems faced by different fields.

Term	Description
Black-Box Models	A machine learning model that is difficult, if not impossible, to interpret. In particular, in the case of regression and classification problems, where the function learned is highly non linear and/or with a multitude of features and parameters. Examples are Deep Neural Network, Tree Ensembles and Support Vector Machines with specific kernels (such as RBF or polynomial with high degree kernels).
Transparent Models	Generically used with the opposite meaning of black-box models, it indicates a model that is interpretable by design. Barriedo Arrieta et al.[5] indicates three characteristics for a model to be transparent: simulability, decomposability, and algorithmic transparency. The first refers to the ability of the model to be simulated by a human, the second that the model can be easily decomposed into smaller pieces, and the latter indicates the ability of the user to reproduce the process followed by the model, starting from the data and ending with the output.
Interpretable Artificial Intelligence	Even though it is usually used as synonymous for Explainable Artificial Intelligence, the majority of the times it is used to refer to systems (or models in case of machine learning) that are intrinsically interpretable, such as Linear Regression, Logistic regression, and GAM among others.
Explainable Artificial Intelligence	Probably the most common term used in the literature and also preferred from some companies[a]. In combination with Explainable Machine Learning is used to refer to all the methods and techniques that enable the final user to get explanations from a black-box model. Thus, it usually includes all the techniques used as a proxy to explain methods that are not intrinsically interpretable. Summing up, a post-hoc analysis of a black-box model.
Responsible Artificial Intelligence	Responsible AI is a branch of AI that takes into account social values as fairness, moral values, and ethical consideration. The key principles of interest in this field can be summarized with the acronym F.A.T., Fairness, Accountability, and Transparency, which emphasizes the focus over the social aspects on socio-technical systems. Given the main principles of the field, responsible AI results in a very interdisciplinary field, bringing together various professionals, computer and social scientists among others, to investigate technical systems that make a big impact to the society.

Table 2.1: Table to summarize the taxonomy used in this paper related to Explainable Artificial Intelligence and other related fields.

[a]For example see the white paper "AI Explanations Whitepaper" available at https://cloud.google.com/explainable-ai from Google LLC (visited on February 16th 2021)

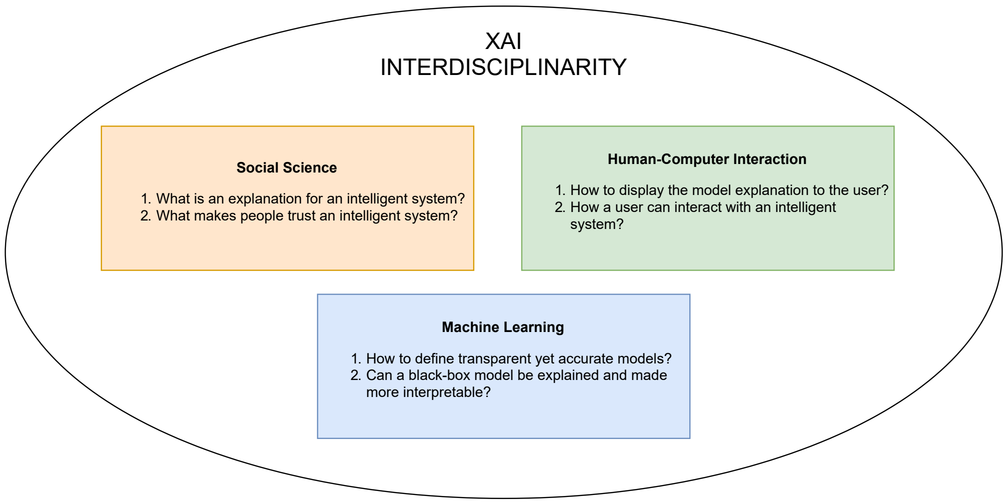

XAI
INTERDISCIPLINARITY
Social Science
1. What is an explanation for an intelligent system?
2. What makes people trust an intelligent system?
Human-Computer Interaction
1. How to display the model explanation to the user?
2. How a user can interact with an intelligent system?
Machine Learning
1. How to define transparent yet accurate models?
2. Can a black-box model be explained and made more interpretable?

Figure 2.2: General schema of the communities involved in the development of XAI, namely Social Science, Human-Computer Interaction, and Machine Learning communities. The key research questions are highlighted under each field.

2.2.1 Social Science

What is an explanation? XAI is deeply related to social science since the path of artificial intelligence is now and could be even more in the following years, strictly related to the development of human society, with a lot of new challenges that have to be faced and possibly radical changes our everyday lives. Sometimes forgotten by the more algorithmic-centered research, humans decision and intervention have a big impact on a large part of AI models and are an essential part of their function. Thus, the main questions posed by social science connected with the XAI world are related to the fundaments of human reasoning, such as what is an explanation, or how an intelligent system can explain its work to a human being, or what is the purpose of explanation in the causal reasoning. An example of this kind of work is the series written by Hoffman et al.[26][27][31] where the leitmotif of their work is the search over the motivations behind the causal reasoning in the human begins, and, in particular, what can be seen as causal reasoning in intelligent systems. Thoughts about this kind of problem are essential to answer a very simple but fundamental question: what do we expect as an explanation from an intelligent system? That is clearly related to: which kind of explanation do we expect from an XAI system to a person in flesh and bones?

How to gain trust Another key question that social science is tackling is the trust in AI, which is different from the one that people normally have with other technologies that are normally used. The missing of trust in new technology can have multiple impacts on its development, and it is inevitably linked with the speed of automation of certain social task[49]. A way to gain more trust is generally obtained with a more transparent AI system, starting from the assumption that the more transparent a system is, the more accurate and fair must be. This assumption is not always satisfied, because in a certain domain a more transparent design can bring less accuracy and possibly breaches for offenders to be exploited and gain advantage from it[53].

2.2.2 Human-Computer Interaction

Present an explanation Related to the HCI field, new visual analytics tools are now being proposed to help researchers fully understand specific machine learning models, an interesting example of this is the thread published on Distill by Cammarata et al. [9] in which a "zoom-in" over the Inception v1 neural net is proposed, i.e. a deep inspection of each layer of that particular deep artificial neural network used to get insights of what the neurons and the circuits they form have learned. Other than this example, new visual techniques can help not only developers and engineers but also final users and decision makers to gain insights and understand what the system has learned.

Interaction with AI The advance of XAI systems is also tightly associated with the development of a new way of interaction with them. Different studies have been done on the impact of human interaction during the learning process of a system, showing that we might want to rethink the learning algorithm and to empower the user to dramatically change it[2]. To do that, new interfaces have to be created and overall control of the system has to be achieved to gain the possibility to solve possible behavior mismatch. Therefore, explainable systems must be developed to permit full control, concerning black-boxes where there are very few possibilities of management.

2.2.3 Machine Learning

Transparent and accurate models The machine learning community has basically approached the need for more intelligible models in two ways. The first one has been to developing numerous techniques trying to create intelligible yet accurate models. Numerous advancements have been made over traditional techniques such as logistic regression, Generalized Additive Models (GAM), and mixed models among others.

Explanations for black-boxes The second approach focused on the development of post-hoc explanations for black-box models. A new approach to this kind of technique is the scope of this thesis. Thus, in the following sections, all the techniques and related works presented belong to this specific part of the research community. However, even though the landscape of this thesis is limited to a particular view, it is worth mentioning that XAI is intrinsically a multidisciplinary subject and there is the need for a holistic evaluation considering all its facets.

2.3 Motivations and skepticism

In this thesis we focus on approaches to explain black-boxes, in particular tree ensembles. Trying to make black-box ML models more transparent can be seen as a meaningless operation, since one can think that the major task of a model is optimizing some performance measures, such as accuracy, without caring too much about what is the function learned and how it works under the hood. On the contrary, nowadays it seems indisputably urgent to investigate over black-box models to explain their decision to answer to, at least, four main questions:

1. Can we justify a decision of a black-box model?

2. Can we gain more control from a black-box model?

3. Can we effectively improve the accuracy of a black-box model with an accurate debugging?

4. Can we discover new knowledge from a black-box model?

It is also true, on the other hand, that we might want to prefer an interpretable model instead of using a post-hoc explanation when it is part of a high-stakes decision-making system. Motivation and skepticism about this approach are presented below.

2.3.1 Main motivations

Four main motivations have been provided by Adadi and Berrad, in a survey over the latest development in XAI[1], they are specifically: explain to justify, explain to control, explain to improve, and explain to discover. In the paragraphs below, they are presented and associated with some real scenarios that can be useful to contextualize the arguments that support the need to explain black-box machine learning models.

Explain to justify The need for explanation to justify a certain decision is essential to ensure that a certain decision or a certain outcome from an AI framework has been made as if it was made by a rational human decision, i.e. we need the assurance that a particular decision has been made with a sensible and intelligible process that is supported by the data. An explanation to justify a particular output from a model can possibly be used also in some legal scenario where the "right of explanation" applies and to justify a possible unwanted output, e.g. an avoidable accident in a self-driving car or an unfair negation of a loan for a certain type of person.

Explain to control Explain a certain AI model is also an attempt to get better control over the whole system. Being able to explain the details of a certain model can help us to prevent possible failures, speed up the resolution of possible implementation bugs, and possibly avoid adversarial attacks.

Explain to improve Besides the possibility to get better control of a model, its explainability can help the researchers to improve quickly its performance. This is due to the fact that all the improvements can be focused on some particular aspects of the model without doing a blind search and trying to improve it without knowing the impact of the modifications.

Explain to discover A model explanation could be very useful to discover new rules. If the modern machine learning techniques can be used without domain expertise, it means that in some way the model builds a set of rules that describe the problem; therefore we can retrieve these rules and possibly broader the human knowledge.

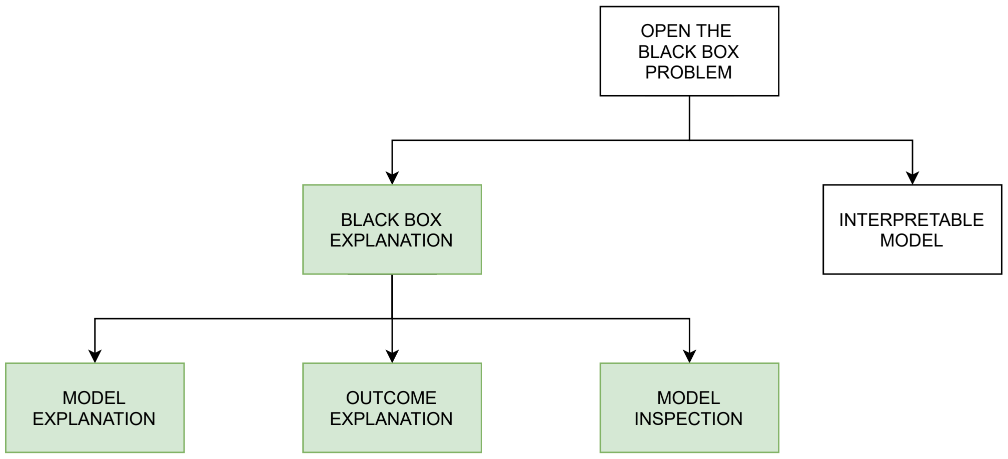


Figure 2.3: A possible XAI taxonomy proposed by Guidotti et al.[23] Highlighted in green the part of interest of this thesis: the black box explanation problem.

2.3.2 Skepticisms

In the previous paragraphs, we have described the major motivations to adopt techniques to make black-boxes interpretable via XAI techniques, however, even though they are all plausible and sensible, we might argue if all this is really necessary. As Rudin has presented in one of her papers [44], we might want to use an interpretable model first (if possible) and then use a black-box model (only if needed) with an explanation. This assertion is especially true thinking about the typical tradeoff depicted between learning performance and effectiveness of explanation, in fact, we cannot really generalize this concept to all the problems and all the datasets. Various techniques for knowledge discovery and data mining can lead to incredible performance for machine learning algorithms without bothering complicated models. Therefore, when it is possible and we can get good performance measurements with an interpretable model, it is better to use it without trying a black-box model and then trying to explain its behavior. This is true except for the fact that one day the block-box model can became easily interpretable with a consistent, accurate, and well-tested explanation. When this happens, we can say that the explanation can become part of the model and the model can be "promoted" to the class of interpretable model. Other than the fact that we might want to prefer an Interpretable Model to a black-box explained, there are other open problems that we might want to consider before applying these techniques: at the moment of writing, the most applied post-hoc explanations techniques are not so reliable, as showed by Slack et al.[50] with examples of adversarial attacks on explanations provided by LIME and SHAP (presented in the following chapter). These limitations, are, among others, to be taken into consideration when a new technique is created.

2.4 Type of explanations

Following the taxonomy proposed by Guidotti et al.[23], and illustrated in Figure 2.3, the type of explanation that can be used are divided into three big groups based on their explanation goal: model explanation, outcome explanation, and model inspection. To achieve these explanation goals various techniques have been adopted through the years for a variety of black box problem, and for tree ensembles they include: tree prototyping, feature importance analysis and visual analysis. In this section a description of the categorization listed above is presented.

2.4.1 Categorization by explanation goal

Giving an explanation of something is not always obvious and is also not so clear what we mean by explanation (as also described in subsection 2.2.1). That said, explanation techniques can be divided into three categories formally described by their goal as follows.

Model explanation Inside the model explanation group, there are all the techniques that create an explanation starting from the model to be explained and the dataset used. This explanation is usually achieved through an interpretable model trying to depict the behavior of the original black-box model. The formal definition is the following.

Definition 5 (Model explanation problem)**.** Given a model to be explained b and a dataset D, the *model explanation problem* consists in finding an explanation $e \in E$, where E is a human interpretable domain, that explains b, and e is usually retrieved by global predictor c_g created in function of the model to be explained b and the dataset D, $c_g = f(b, D)$.

When a interpretable global predictor c_g has been found, the explanation e can be retrieved through an explanation logic ϵ_g from c_g and X, $e = \epsilon_g(c_g, D))$.

Outcome explanation Starting from a model we might not want to explain the whole logic behind it, but we might want to describe only the process that has followed to come up with a particular outcome, this is called *outcome explanation problem.*

Definition 6 (Outcome explanation problem)**.** Given a model to be explained b and a new sample x, the *output explanation problem* consists in finding an explanation $e \in E$, where E is a human interpretable domain, that explains b the outcome of b over the sample x.

This process can pass through an interpretable local predictor c_l in an analogous way as a global predictor c_g can be used in the model explanation problem.

Model inspection The last form of explanation that can be described is the *model inspection problem*, where the explanation is provided with a visual or textual representation in order to understand some specific properties of the black box model or of its prediction.

Definition 7 (Model inspection problem)**.** Given a model to be explained b and a set of instances X, the *model inspection problem* consists in finding a visual or textual representation $r = f(b, X)$ of some property of b.

The main difference between the model inspection problem and the model explanation problem is that the latter provide an explanation of the system as a whole, while the former is specific to some properties of the model to be explained.

2.4.2 Explanation Techniques for Ensemble Methods

In order to answer to answer to the black box explanation problem multiple types of techniques have been studied. In particular, with respect to the explanation for tree ensemble methods, we can identify 3 major categories: tree prototyping, feature interaction analysis, and visual analysis.

Tree prototyping Tree prototyping is usually used to answer the model explanation problem. In this case, the global predictor c_g is a decision tree, that is able to mimic the behavior of the tree ensemble. When the tree c_g is constructed, it is fairly easy to extract an explanation through an explanation logic since a decision tree is fairly interpretable under a certain size. This technique can be seen as a form of summarization of the ensemble to a single tree, or a small set of trees. The advantages of this method are that there are a lot of good metrics and heuristics to create this explanation, but the main problem is that most of the time the accuracy of the prototype created is well below the one of the ensemble, and sometimes leaving uncovered some possibly relevant less frequent cases. In addition, it is worth mentioning that more than one prototype can be used to explain the initial model.

Feature Interaction Analysis Probably the most popular method used to explain a tree ensemble, it has been used to answer the outcome explanation problem. This type of technique is based on the study of the interaction between the features and their contribution to the output of a model given a specific sample. The interpretable local predictor c_l is usually a simple description of the contribution of each feature to the output. The advantage of this technique is that it can be very useful to analyze misclassifications and possible bugs associated with a prediction on a particular sample in a very effective way. One of the major drawbacks of this technique is that is designed to explain only a small portion of the model (it is in fact sometimes also called local explanation), thus to get a better understanding of the black-box model it is usually accompanied with a global model explanation.

Visual Inspection Inside the visual inspection techniques there are all the methods used to answer the model inspection problem. Two methods are common inside this category, namely the Partial Dependence Plot(PDO) and plot to visualize sensitivity analysis. Visual inspections methods can be very useful to explain single predictions, but they are generally not so accurate to explain the behavior of the entire model due to possible visual clutter when the number of features or samples is very large, or due to their over-simplified visualization, when one or two features are considered over hundreds.

2.5 Evaluation methods

Different approaches have been used to evaluate the explanation techniques developed, however at the moment there are no common benchmarks available to measure the goodness of these methods[25]. In fact, the evaluations usually used for the explanation are based on user studies that measure their effectiveness, but these metrics are often project-dependent. To give an idea of a possible framework to evaluate different XAI systems, we can mention the one used for the DARPA's

XAI program[24] to evaluate different projects. In this program two main evaluation categories were used: ML model performance and explanation effectiveness. The former was a set of measurements related to the accuracy/performance of the explainable system created versus the not-explainable baseline. The latter was instead a collection of measurements based on user case studies, such as the improvement of the user performing the task for which the system was designed and the users' subjective ratings for the quality of explanation.

2.6 Conclusion and summary

Conclusion In this chapter, we have presented an overview of the XAI world, which is a very emerging field and aims to emphasize the focus on explainability and interpretability of an intelligent system. This open-up a lot of related research questions from different disciplinary areas, among which we can highlight the social science, the HCI, and the ML area. In this thesis, we focus on the machine learning facade of XAI and we try to explore a possible way to solve the model explanation for tree ensembles through a well-known technique borrowed from the data mining field, which is frequent subtree mining. This type of approach share almost no link with the already present solution in the literature to the best of our knowledge and should be considered primarily as an investigation for possible future research.

Summary The summary of the chapter divided by sections is the following:

- In section 2.1 we have reviewed the definition of common terminology in the XAI field.
- In section 2.2 we have presented the research focus of the main communities involved in the development of the XAI field, namely the social science, the HCI, and the ML area.
- In section 2.3 we have illustrated the motivations to make AI more explainable, and the need for techniques and framework to achieve it. On the other hand, we have discussed also the main limitations of this type of approach and what has to be taken into account when an explanation is used.
- In section 2.4 we have illustrated a possible categorization of the explanation techniques for their purpose in three categories: model explanation, outcome explanation, and model inspection. We have also presented a categorization for the main types of explanation for tree ensembles, namely tree prototyping, feature interaction analysis, and visual inspection.
- In section 2.5 we have discussed the lack of a common benchmark for the techniques developed to explain a black-box. However, we have briefly presented a benchmark to exemplify the concept.

Chapter 3

State of the art

In this thesis, since we are interested in tree ensembles, we review the main explanation methods already proposed for this type of ML model. Summarizing the works presented in the literature, we can identify three principal categories of explanations: tree prototyping, feature interaction analysis, and visual inspection. For each category, a small subset of works has been chosen as representative based on their peculiarities, i.e. if they provide a strongly different approach from other methods, on their impact on the research community, and the availability of a open-source implementation. When an open-source implementation was available, for illustrative purposes, it has been tested over an ensemble of decision tree created with the Light Gradient Boosting Machine (LGBM) package for Python[29]. The dataset used to conduct the tests was the Bike Sharing Dataset downloadable from the University of California Irvine (UCI) Machine Learning Repository and made available from Fanee-t and Gama[17].
In this chapter, we first review the main characteristics of the dataset used to conduct the tests specifying its properties. Then, for each category of explanation, we list the selected methods and highlight the advantages and disadvantages of each.

3.1 Illustrative Dataset and Model

The Bike Sharing Dataset is available with two granularities, daily or hourly, in this case it has been chosen the daily granularity. That means that the dataset contains for each day between the years 2011 and 2012 the daily count of rental bikes in the Capital (a company based in Washington D.C.) bike-share system, associated with weather and seasonal information. For each sample in the dataset 12 features have been used as presented in Table 3.1.
The goal in this problem is to predict the number of rental bikes *cnt*, and for an illustrative purpose a simple LGBM Regressor model has been fitted, on intent without changing any default parameters provided by the library and without any preprocessing of the initial dataset to find possible flaws or bias in the model.

3.2 Tree prototyping

Tree prototyping can be seen as a very natural way to explain an ensemble of trees and its development date back to the late '90s. It consists of creating one or more trees that summarize the entire ensemble. This is a sensible explanation method

Feature Name	Description
instant	The sample progressive index. It is equal to 1 for the first day and 731 for the last record.
season	An index that corresponds to a particular season (1: winter, 2: spring, 3: summer, 4: fall).
yr	The year when the sample has been recorded.
mnth	The month when the sample has been recorded.
holiday	A boolean feature to indicate if the day was a holiday or not.
weekday	The day of the week.
workingday	A boolean feature to indicate if day was neither weekend nor holiday.
weathersit	An index between 1 to 4 indicating the weather condition. 1 corresponds to prefect weather to cycling, 4 to heavy rain, ice pallets, etc.
temp	The normalized temperature in celsius. The values t_{norm} of this feature has been computed as $t_{norm} = \frac{t-t_{min}}{t_{max}-t_{min}}$, where t is the raw value of the feature, $t_{min} = -8$, and $t_{max} = 39$.
hum	The normalized humidity value, from 0 to 0.9725.
windspeed	The normalized wind speed, from 0.022392 to 0.507463.
cnt	The count of the total rental bikes. This is the label used for the regression problem.

Table 3.1: Features of the Bike Sharing Dataset used to show the solution proposed by each explanation method analyzed.

because a single tree with an adequate depth and number of leaves is simple to be interpreted. Tree prototyping due to its affinity with tree ensemble methods is one of the most discussed approaches to explain this type of black-box, and in this section, we review five methods that can be considered representative for this explanation approach. It is worth noticing that even though a lot of works are present in the literature related to this particular topic, very few implementations are publicly available and its usage seems now limited. A summary of the tree prototyping techniques presented is proposed at the end of the section in Table 3.3.

Tree metrics and clustering One of the first attempts to find an explanation for tree ensembles via tree prototyping focused on finding metrics of similarity between trees inside the ensembles[12]. In this case the solution is based on the fact that multiple trees inside the ensemble are similar, therefore multiple metrics could be defined between trees to compute their similarity. The final goal of the similarity computation between trees is to cluster them and to find a representative tree for each cluster explaining the tree ensemble with a series of tree prototypes. In particular, the authors proposed three metrics based called *fit metric*, *partition metric*, and *tree metric*. The former is a metric over the predicted values of the training set, the second is a measure of the distance between trees that is based on the difference in the partition of the training set in the two trees considered, and the latter account for the topology of the trees. For example, as fit metric they propose $d(T_1, T_2) = \frac{1}{N}\sum_{i=1}^{n}(\hat{y}_{i,1} - \hat{y}_{i,2})^2$ for regression trees, where T_1 and T_2 are the two trees considered, $\hat{y}_{i,1}$ and $\hat{y}_{i,2}$ are the predictions for the i-th element of the first and second tree respectively. Finally, when the clusters are created with

a standard hierarchical clustering technique, the tree with the highest likelihood among the cluster is chosen as representative.
This method worth to be mentioned because it can be easy to be implemented and it shows that doing clustering over the trees that compose the ensemble can provide insights about the black-box. However, it is not so clear how to choose the number of clusters, and how to combine the information retrieved by the various representative trees to explain the whole model.

Tree prototyping from class distribution In 2007 Van Assche and Blockeel[52] presented a method to learn a single decision tree exploiting the class distribution estimate provided by the different trees that belong to the ensemble. Specifically, in their work they highlight that the exact function learned by a tree ensemble can be expressed as a single tree using up to 2^d leaves, where d is the number of different tests in the ensemble, making it too large and not so interpretable in most of the cases. Thus, they create a new tree using the information gain based on an estimation of the class distribution over the ensemble as the splitting criterion. The class distribution is estimated both via the information retrieved by leaves and with the training dataset. This procedure permits to add to the prototype only the splits that are "most informative" for the ensemble. To further reduce the size of the tree produced, prepruning and a "preserving labeling" stopping criterion are employed. The first refers to a pruning technique during the learning phase based on a class probability threshold to be maintained in every node. The second means that no further splits are made when all the samples associated with a node are classified within the same class by the ensemble.
The evaluation showed that the accuracy of this method was most of the time lower than the ensemble used but slightly better than other methods. However, the evaluation over the stability of this tree prototyping method showed that the tree created was not so stable compared to the ensemble from which it derived, where the stability is "the probability that models generated by the same learner on different training sets will give the same prediction to a randomly selected instance". The lack of stability is common with the majority of the other tree prototyping methods.

Combining Multiple Models Differently from the methods described above is Combining Multiple Models (CMM)[15], that instead of directly investigate the ensemble, uses the information retrieved from its predictions to recover the implicit decision regions of the model. The rationale behind this technique is to augment the dataset used to train the decision tree prototype by adding a large number of examples generated and classified according to the black-box. It can be seen as a meta-learning, where the new interpretable learner is created from a dataset integrated with the prediction from the tree ensemble. Even though the general framework can be easily implemented and understood, an important consideration has to be made, as also emphasized by the authors. During the meta-learning phase, the additional samples introduced in the new dataset have to follow the sample distribution of the initial training set. This is due to the fact that many ML algorithms are sensitive to the sample distribution and using a poor sampling algorithm can greatly impact the accuracy and the relation with the representation of the "true" decision region, i.e. the distance from the true $f(\mathbf{x})$ that resides behind the data and the approximate function learned $\hat{f}(\mathbf{x})$.

Partition Aware Local Model Khrisnan and Wu[32] proposed an agnostic technique to summarize every black-box model with a two-part surrogate model, namely Partition Aware Local Model (PALM). In their work, they create a meta-model that represents partitions of the training data and, in each partition, they mimic the behavior of the black-box with an arbitrary complex model. This approach is not really a tree prototyping method, but it creates a decision tree as a surrogate model to subdivide the initial dataset into k partitions that "best explain" the prediction of the black-box over the training dataset. These partitions are made solving an optimization problem to find k parametrized probability distributions, with an algorithm very similar to K-Means, where instead of recomputing the cluster center at each iteration, a gradient ascent approach is employed. After having found the k partitions, the training set is projected into a subspace of features $\mathcal{X}_e$ that are considered "explainable" and associated with one of the k partition created, forming a new dataset D_e. At this point, a decision tree is trained to classify the samples in the new dataset to one of each k partitions. Finally, for each k partitions, arbitrary complex models are created to mimic the black-box predictor. Thus, this method provides an agnostic explanation that tries to be either simpler with respect to the original model, using a partition of the data, and also accurate like the black-box, using arbitrary complex models in each partition. However, it seems not so obvious how to choose the parametric distribution to find the k partitions and how to find the best k to explain the model without losing too much accuracy.

Table 3.2: Explanations via tree prototyping

Name	Ref.	Year	Short description
Tree Metrics	Chipman et al.[12]	1998	They authors define various metrics on trees, and describe how to cluster them with the aim of finding archetypes that best explain the whole model.
CCM	Domingos[15]	1998	CCM creates a new interpretable model, such as a decision tree, with a dataset augmented by the prediction of the initial ensemble method used.
-	Van Assche and Blockeel[52]	2007	The authors provide a new method to summarize the ensemble with a single decision tree where each split is based on the class distribution retrieved from the ensemble and from the training set.
PALM	Krishnan and Wu[32]	2017	PALM creates a tree to partition the initial dataset and then it learns a simpler model inside each partition. Everything is created to mimic the prediction of the black-box method over the initial dataset.

Table 3.3: Methods for tree prototyping that can be used to explain tree ensemble methods, listed in chronological order.

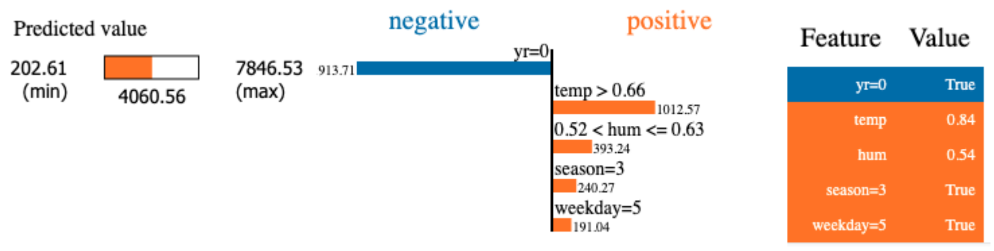


Figure 3.1: A lime explanation for a sample of the illustrative dataset. The numbers under the columns *negative* and *positive* indicates the weights of the feature in the linear model c_l created by LIME. In this case, the explanation says that the year has a strong negative impact on the predicted value. On the other hand, an over the average temperature, a normal level of humidity, and the summer season have a positive impact on the prediction.

3.3 Feature interaction analysis

Another useful approach to explain a tree ensemble is the use of *feature interaction analysis*. In this thesis, we use the term feature interaction analysis to indicate all the methods that study the impact of the features on a black-box model. There are different methods described in the literature that follow from different well-known approaches, such as sensitivity analysis, and game theory among others. In this section, we review some of the relevant techniques for this type of explanation that are normally used to answer the outcome explanation problem. A summary of the features interaction explanation techniques presented is proposed at the end of the section in Table 3.4.

Global Sensitivity Analysis One of the first example of feature interaction analysis is Global Sensitivity Analysis (GSA)[13]. In this work, the authors suggest that sensitivity analysis can be used to explain the feature interaction of a black-box. With this intent, they generalize other previous studies made on one and two dimensional (1D - 2D) sensitivity analysis and present an algorithm to analyze the response of a black-box under the variation of 1 up to M feature, where M is the number of features used from the black-box. In the MD sensitivity analysis a search matrix of input is created selecting L values for each feature in order to test the prediction of the black-box on a large part of the sample space. After having computed all the predictions in the search space, various metrics can be used to quantify the sensitivity of the black-box from variations of the M features selected, such as *range*, *gradient*, and *variance*[30]. Given its specifications, the method can be used on every black-box model and not only on tree ensemble and can give good insights also easy to visualize and to be interpreted. However, the main problem is that the scan over M dimension results infeasible when M is really large, or the number of input values L increase. This can be mitigated selecting a reasonable number of features and input values, however no effective strategies have been proposed to pursuit this goal.

Local Interpretable Model-agnostic Explanations With a focus on the outcome explanation problem, Ribeiro et al.[43] present an agnostic method to explain the prediction of any black-box, they call it Local Interpretable Model-agnostic Explanations (LIME). LIME samples points from the neighborhood of

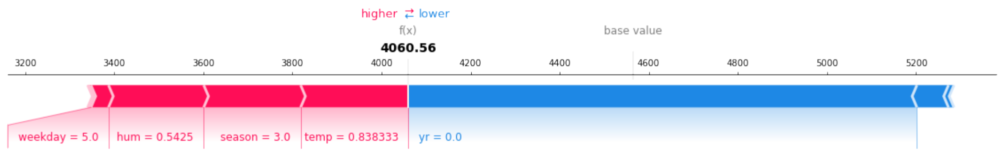


Figure 3.2: SHAP local explanation for the same sample as in Figure 3.1. In this case, the contribution of the season feature is stronger than the one representing the humidity level. However, except for this little discrepancy, the two explanations are very similar.

the point under investigation and locally approximates the results with an interpretable model, such as a decision rule or a linear model. In LIME, the problem of finding an explanation $e(b, x)$ of a black-box b and a sample x is defined as an optimization problem:

$$e(b, x) = \underset{c_l \in C}{\operatorname{argmin}} \mathcal{L}\left(b, c_l, \pi_x\right) + \Omega(c_l) \tag{3.1}$$

where c_l is the local predictor (as defined in Definition 6) derived from a class of function C, $\mathcal{L}\left(b, c_l, \pi_x\right)$ is a measure of *unfaithfulness* of c_l in the neighborhood π_x of x, and $\Omega(c_l)$ is the complexity of the model. It is worth noticing that the neighborhood π_x could also include points that do not belong to the original dataset. The authors, in their implementation, used the class of linear model as C, and the square loss in π_x as a measure of faithfulness. An example of this type of explanation with a linear model is presented in Figure 3.1 over the illustrative model.
In addition, it is also presented a method to choose k instances to solve an optimization problem to find the maximum coverage of the input space, trying to solve the model explanation problem through k outcome explanation.
This local explanation method can generally approximate well the prediction of a black box in a neighborhood of the sample considered. However, it is not always easy to define the right neighborhood of the sample and choose the right class of function C in advance.

TreeExplainer Finally, Lundberg et al. [35] present TreeExplainer, a method to compute *optimal* local explanation based on the on the classic game-theoretic *Shapley Values*[48] to represent the feature importance in every prediction of a model. In this cooperative game theory, each member of a coalition should receive a fair payoff given proportional to their marginal contribution. The bridge between theory and practice is easily created: each feature is a member of the coalition and its contribution is the feature's importance on a specific prediction. In particular, the computation of the Shapley values proposed in this explanation method is based on a feature additive attribution method, i.e. a linear model. Thus, to emphasize that the Shapley values have been computed in an additive way, they have been called SHapley Additive exPlanation (SHAP) values[34]. Specifically, for each prediction, the explanation model c_l is a linear model defined as[38]:

$$f(\mathbf{x}) = c_l\left(\mathbf{x}'\right) = \phi_0 + \sum_{j=1}^{M} \phi_j \mathbf{x}'_j$$

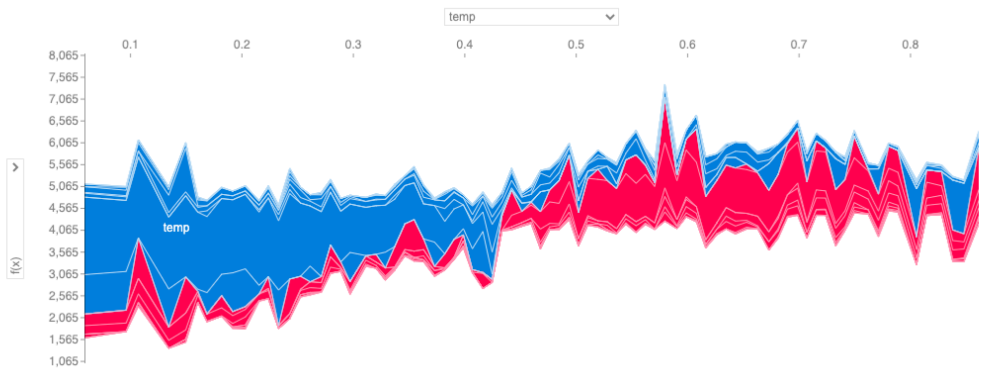


Figure 3.3: SHAP global explanation for the temperature feature. In this type of global explanation, the plot is created by stacking every feature importance contribution calculated for each sample in the training set. As in Figure 3.2, the blue color means a negative impact with respect to the average value, whereas red means a positive impact. In this case, it easy to see that the lower the temperature, the lower is the predicted number of bikes rented.

where $\phi_0 = E_X(\hat{f}(\mathbf{x}))$ is the average prediction, $\mathbf{x}' \in \{0,1\}^M$ is the coalition vector, where a 1 means that the feature is present in the coalition and a 0 that is absent, and ϕ_j are the Shapley values. Based on this idea the author present a model agnostic approximation for their computation called KernelSHAP and a fast polynomial computation for tree ensembles called TreeExplainer. In figure Figure 3.2 is presented a prediction explanation for the illustrative model.
In addition, various techniques for the model explanation problem have been proposed. One of these is a global representation of the impact of a single feature on the entire dataset created by stacking the predictions for each training sample. An example is presented in Figure 3.3.
Summing up, SHAP has a solid theoretical foundation on game theory and can be computed quickly on tree ensembles. However, computing the Shapley values is generally slow and when the TreeShap technique is used it can produce unintuitive feature attributions[38]. SHAP has been also recently showed not so robust against adversarial attacks performed to hide the bias present in the model to be explained[50].

3.4 Visual inspection

A widespread approach used to explain black-box model and thus also a tree ensemble is the use of visualization techniques. In this section we briefly present three major techniques of explanation via visual inspection used with three ensembles, namely: Partial Dependence Plot, Individual Conditional Expectation, and Accumulated Local Effect. A summary of the visual inspection techniques presented is proposed at the end of the section in Table 3.5.

Partial Dependece Plot Friedman, in 1999, in the same paper where it describes the gradient boosting framework[19] it also propose the Partial Dependence Plot (PDP) as one way to interpret the relation between the outcome of a Gradient Boosting Machine and a subset of features of the training dataset. Even tough the method was initially described for a gradient boosting machine it is

Name	Ref.	Year	Short description
GSA	Cortez and Embrechts [13]	2011	GSA is an agnostic method that applies sensitivity analysis to the features used from the black-box.
LIME	Ribeiro et al. [43]	2016	LIME tackles the outcome explanation problem with an agnostic method that approximate a prediction with an interpretable model in the neighborhood of the sample.
TreeExplainer	Lundberg [35]	2020	TreeExplainer is a game theoretic approach (based on Shapley values) to measure the feature importance of a certain prediction. Global model interpretation have also been proposed starting from a collection of local explanations.

Table 3.4: Methods for feature interaction analysis that can be used over tree ensembles, listed in chronological order.

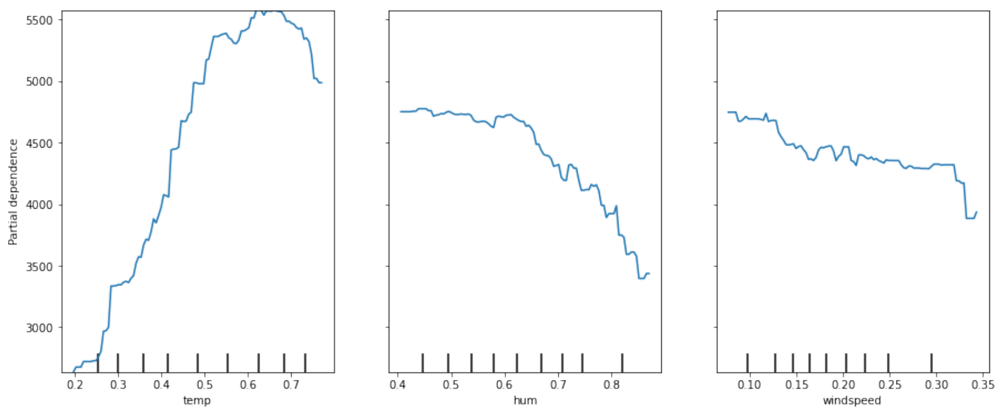


Figure 3.4: Partial Dependence Plots, created with scikit-learn[39], over three numerical variables in the illustrative model, namely temperature, humidity and wind speed.

applicable to every tree ensemble method. This is based on the fact that the input variable $\mathbf{x}$ can be divided in two subsets: the target subset $\mathbf{x}_T$ and its compliment $\mathbf{x}_C$ s.t. $\mathbf{x}_T \cup \mathbf{x}_C = \mathbf{x}$, thus the estimated function $\hat{f}(\mathbf{x})$ can be expressed as $\hat{f}(\mathbf{x}) = \hat{f}(\mathbf{x}_T, \mathbf{x}_C)$. Given that, we can analyze the relation between the output and the set of target features $\mathbf{x}_T$, as the marginal expected value of the function $\hat{f}(\mathbf{x}_T, \mathbf{x}_C)$ knowing the probability of $\mathbf{x}_C$. Therefore, the partial dependence function is defined as:

$$\bar{f}_T(\mathbf{x}_T) = E_{\mathbf{x}_C}\left[\hat{f}(\mathbf{x}_T, \mathbf{x}_C)\right] = \int \hat{f}(\mathbf{x}_T, \mathbf{x}_C) d\mathbb{P}\left(\mathbf{x}_C\right)$$

Naturally, we cannot compute $\bar{f}_t(\mathbf{x}_T)$, but it is instead estimated by computing the average prediction in the training data, therefore the practical definition become:

$$\bar{f}_T(\mathbf{x}_T) = \frac{1}{N}\sum_{i=1}^{N} \hat{f}\left(\mathbf{x}_T, \mathbf{x}_C^{(i)}\right)$$

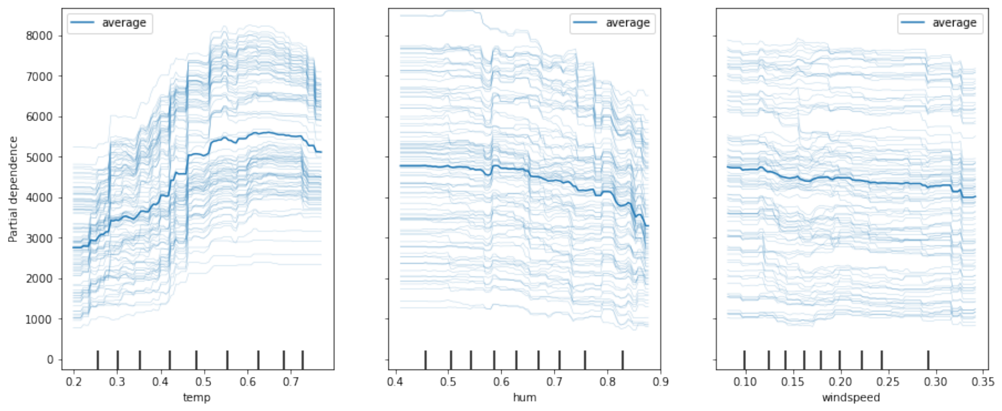


Figure 3.5: Individual Conditional Expectation over the same three numerical variables in Figure 3.4, with 100 sample.

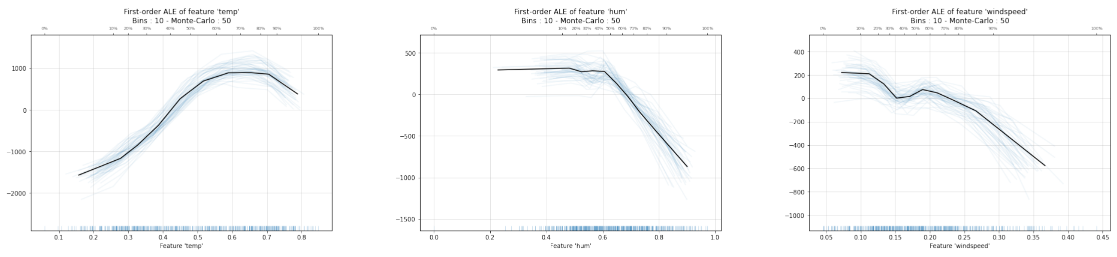


Figure 3.6: Accumulated Local Effect for the feature *temp*, *hum*, and *windspeed*, as done for the other visual inspection methods. In this case on the vertical axis we have the difference from the average prediction.

The main advantage of this method is that is simple to implement and can give a good overview over one or two features. However, it is practically impossible to plot a PDP with more than two features in the target subset $\mathbf{x}_T$ and this method can give a misleading overview if two or more features are correlated with each other. Examples of PDP plots created from the illustrative model are shown in Figure 3.4.

Individual Conditional Expectation To enhance the PDP plot and avoid to lose too much information averaging the marginal effect of one or more features, in [21] the authors suggest an improved PDP where instead of plotting the average curve, N curves are plotted, one for each entry in the dataset. This give to the user the ability to inspect individually each sample and can give more insights about the global modal behavior. However, this can easily led to clutter the visualization if all samples are used. To solve this problem, generally a subsample of the training dataset is used, with a negative impact on the fidelity of the explanation. An example of ICE plot is presented in Figure 3.5.

Accumulated Local Effect Finally, Apley and Zhu recently presented the Accumulated Local Effect method (ALE)[4], trying to solve the problem of PDP when two or more features are correlated. In fact, when in a PDP plot the average marginal effect is computed, the correlation between features in the dataset is not taken into account and unlikely samples are possibly generated. For example, considering the illustrative dataset, we have features that are possibly strongly correlated, such as the temperature *temp* and the season *season*. The problem in

a PDP plot is that during the computation of the PDP functions over the *temp* feature, samples with low temperature during summer and high temperature during winter are created. ALE plots try to solve this problem by calculating the difference in predictions in a specific region of the feature's target space. In this way, the average effect of a feature is computed on the conditional distribution of a feature without mixing the effects of correlated features. When the estimated ALE function $\bar{f}_{ALE,j}(x)$ is computed, it represents the impact of a feature j on the prediction of the sample x. Since in the estimation process the effect is centered, when $\bar{f}_{ALE,j}(x) = 0$ means that the feature j has no effect on the predicted value of x, when $\bar{f}_{ALE,j}(x) > 0$ means that the feature has a positive impact on the prediction, and when $\bar{f}_{ALE,j}(x) < 0$ it means that the feature has a negative impact on the prediction of the label of x. Examples of ALE plots are presented in Figure 3.6.
The main advantage of ALE plots is that they are unbiased by design, and they can greatly deal with correlated features that are the biggest problem with PDP plots. However, their accuracy and their computation time is influenced by the number of intervals selected, and there is no guidelines to choose the right number of division of the feature space.

Name	Ref.	Year	Short description
PDP	Friedman [19]	1999	PDP is a method to show the marginal effect of one or more features on the predictions of the training set.
ICE	Goldstein et al. [21]	2014	ICE is a refinement of the PDP plots, including not only the average marginal effect but also the effect on each sample.
ALE	Apley and Zhu [4]	2020	ALE is an improved version of the PDP plots that tries to solve the problem related to correlated features that is present in PDP plots and consequently in ICE.

Table 3.5: Methods for visual inspection that can be used over tree ensembles, listed in chronological order.

3.5 Conclusion and summary

Conclusion In this chapter, we have review a set of related works to explain a tree ensemble that has been considered relevant for their research impact, or they originality. Following the classification made in section 2.4, we have subdivided the methods in three categories: tree prototyping, feature analysis interaction and visual inspection. We have presented in total 12 methods also with an illustrative example when an open-source implementation was easily available. The method that we propose in our thesis can easily belong to the "feature interaction analysis" category, where the analysis of the relation/interaction between features has been made exploiting the frequent subtree mining techniques. One of the main difference of this approach with respect to the other three that have been described, namely GSA, LIME and TreeExplainer, is that it is not designed to solve primarily the output explanation problem, but it has been created to tackle the model explanation problem.

Summary The summary of the chapter divided by sections is the following:

- In section 3.1 we have presented the illustrative dataset that has been used to make an example of each explanation method where a public and easily accessible, open-source implementation was available. The aforementioned dataset is the Capitol bike-sharing dataset.

- In section 3.2 four techniques have been described that explain a tree ensemble via tree prototyping. In the *tree prototyping*category, we include all the methods that use a single tree to approximate the model learned by a tree ensemble. We have presented two methods that are potentially model agnostic, i.e. they work with all the black-box and two methods that are specific for tree ensemble. The former are CCM and PALM, the latter are Tree metrics and the framework proposed by Van Assche and Blockee (see Table 3.3).

- In section 3.3 we have illustrated three methods, namely GSA, LIME and TreeExplainer, that are used to analyze the feature interaction in a tree ensemble. All the methods taken into account are model agnostic by design, and one has been optimized for the explanation of a tree ensemble, which is TreeExplainer. It is worth noticing that LIME and TreeExplainer can be considered as de facto standard for tree ensembles explanation.

- In section 3.4 we have presented three methods that are commonly used to analyze a tree ensemble and are based on a visual inspection of the plots describing the impact of the features on the model. In particular, we have discussed the partial dependence plot and its enhanced version, ICE. We also considered a recently published method called Accumulated Local Effect (ALE) that tries to solve some well-known problems of partial dependence plots.

Chapter 4

Forest Explanation Through Pattern Discovery

This chapter presents the core of this thesis: an analysis of the potential application of pattern discovery techniques in the field of XAI. In particular, we analyze the potential application of finding frequent subtrees in forests, we discuss how they can represent a particular relationship between features, and we illustrate a possible way to find a specific relationship in a more complex ensemble. We begin the presentation of our project listing the research questions that have led us to undertake the frequent pattern discovery approach to explain a tree ensemble. Then, we briefly recap some methods available in the literature to find frequent subtrees in a forest that we have adopted in our analysis. Moreover, we present an exploratory data analysis of frequent subtrees retrieved in tree ensembles that try to model simple functions, and that suggests that there is a strong relation between frequent subtrees and the function behind the data. At the end of the chapter, a possible framework to generalize the concept and find more complex relationships is presented.

4.1 Research questions

The initial research question that has motivated our research is derived from the fact that all the methods that try to explain an ensemble of trees are rarely focused on the whole model by design, they are instead generally focused on its outcomes or on the dataset from where they are learned. Thus, in this thesis, we focus on the **model explanation problem**. In particular, we highlight that when we want to discover new knowledge, we (as humans) are used to deal with small closed-form expressions that are made up by combining different elementary functions. Specifically, a lot of different research discoveries made in the past has brought to us closed-form expression made up with a reasonable number of variable to explain a large number of different phenomena. For example, we can think about the various closed-form expression in natural science that has been formulated directly from, or confirmed by, raw data. To explain a machine learning model we would like to do the same, thus transforming the knowledge acquired by an ensemble method into an interpretable closed-form expression.

Research Question 1 (Forest explanation as a simple closed-form expression)**.** Can the model explanation problem, as formulated in Definition 6, be solved with an explanation e that is a closed-form expression $f_e(\mathbf{x})$ belonging to a set of arbitrary complex closed-form expression E? If $\hat{f}(\mathbf{x})$ is the function learned by the model we want that $f_e(\mathbf{x}) \approx \hat{f}(\mathbf{x})$.

In order to answer RQ 1 a thorough search of the relevant literature yielded no related article. Therefore, to answer this new research question, the problem has been decomposed in multiple sub-questions that they could be interesting also per se. We hypothesize that particular subtrees in an ensemble could explain a particular type of relation between features, and we want to discover if it is possible to combine the information retrieved by the mining of frequent subtrees to create an arbitrarily complex function.

Research Question 2 (Forest explanation through frequent subtrees)**.** Can we construct $f_e(\mathbf{x})$ as proposed in RQ 1 by using the knowledge of the frequent subtrees present in a tree ensemble?

To solve RQ 2 we highlight that normally a closed-form expression is a composition of functions, therefore, first of all, we want to discover if the model has learned a particular type of relationship between two features and eventually combine all the relations found to create the final $f_e(\mathbf{x})$. For example, if the function learned by a tree ensemble is approximately equal to a sum of the first and the second feature of a sample, i.e. $\hat{f}(\mathbf{x}) \simeq \mathbf{x}_1 + \mathbf{x}_2$, can we find one or multiple tree-based patterns in the model that reflect this relationship? And then, if the function learned by the tree ensemble is $\hat{f}_{comp}(\mathbf{x}) \simeq (\mathbf{x}_1 + \mathbf{x}_2) \cdot \mathbf{x}_3$ and we know which are the frequent subtree when $\hat{f}(\mathbf{x}) \simeq \mathbf{x}_1 + \mathbf{x}_2$, can we exploit this information to find the relation between $\mathbf{x}_1$ and $\mathbf{x}_2$ in $\hat{f}_{comp}(\mathbf{x})$? An affirmative answer to this question could imply that if a tree ensemble has learned a composed function $\hat{f}_{comp}(\mathbf{x})$, we could find each of its sub-components through frequent subtree mining, and then combine them to retrieve the function of interest: $\hat{f}_{comp}(\mathbf{x})$. Therefore, reconsidering the example above, we want to know if there is a link with $\hat{f}(\mathbf{x})$ and the presence of particular subtrees involving $\mathbf{x}_1$ and $\mathbf{x}_2$ in the tree ensemble.

Research Question 3 (Find the relationships between features through pattern discovery)**.** Can we find which type of function a tree ensemble has learned between a pair of features mining its frequent subtrees?

In addition, we want to investigate if this approach can give us the opportunity to improve the accuracy of a tree ensemble adding to its dataset a new feature that represents this interaction. This can be seen as an automatic process for features engineering where we both explain the relation between features and improve the performance of the model under investigation.

Research Question 4 (Improve the accuracy through pattern discovery)**.** Can we improve the accuracy of an ensemble over a dataset adding to the dataset a new feature that represents the feature interaction discovered through its frequent tree-based pattern?

To answer the principal research questions of this thesis, different mining algorithms have been used that are explained in the next section.

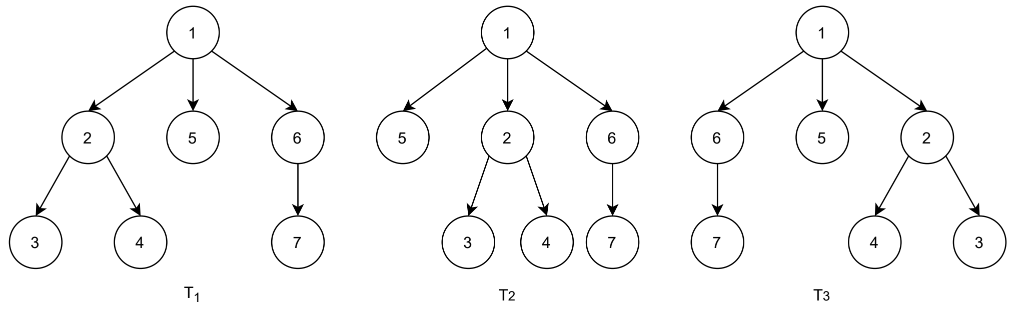


Figure 4.1: Figure with a visual representation of three trees, that can be considered three visualizations of the same **unordered** tree or three different **ordered** trees.

4.2 Frequent subtree mining

With the term *frequent subtree mining* we refer to the procedure of retrieving frequent substructures from a dataset of *labeled* trees, where *labeled* means that at each node v of a tree in the dataset, is associated with a label l. We highlight that in this context every node v is associated with only one label l, but each l can be associated with multiple nodes.

Frequent subtree mining is a well-known field in data mining strictly related to the frequent subgraph mining problem, and that extends techniques from the association rule mining and the frequent pattern mining areas[11].

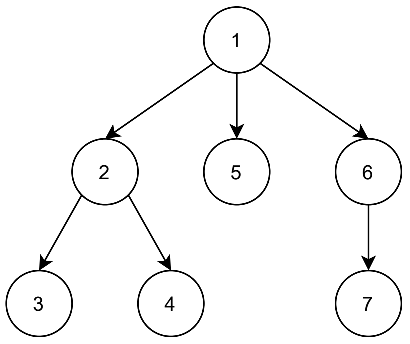
The link between tree ensembles and frequent subtree mining can be made noticing that a tree ensemble can be seen as a dataset of trees after a proper transformation (described in the following section).

In this section we review the mining algorithm used in this thesis trying to answer RQ 3 and RQ 2, starting from the definitions of the different types of subtrees studied.

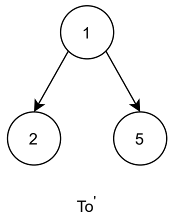
4.2.1 Types of subtrees

Generally speaking, a subtree is a substructure present in a tree and can be defined in a variety of ways depending on the context of use. In particular, two common types of tree substructure are used, namely *induced subtrees*, and *embedded subtrees*. To describe them we add some notation to the one used in section 1.3, defining the *ancestor* and the *descendant* relationship between two nodes v_1 and v_2, the difference between *ordered* and *unordered* trees, and the *preorder* function. It is also worth to notice that we are generalizing the definition of a tree data structure, presented in section 1.3, because in the frequent subtree mining context a tree is an acyclic, connected, **labeled**, and directed graph $G = (V, E)$.

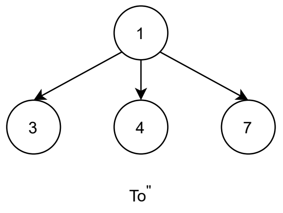
Ancestor and descendant With the notation $v_1 = ancestor(v_2)$ we indicate the fact that there exists a path from the node v_1 to the node v_2 in a tree T. In that case, we say that v_1 is an ancestor of v_2 and v_2 is a descendant of v_1. For example in the tree T_1 in Figure 4.1, the node v_1 labeled with a one, is an ancestor of the node v_3 labeled with a three, thus, in this case, $v_1 = ancestor(v_3)$ and v_3 is a descendant for v_1.



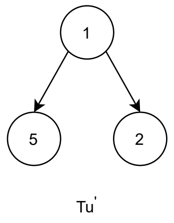
(a) Initial tree T



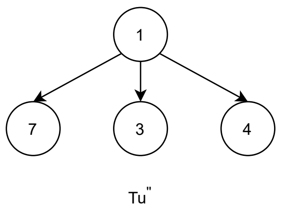
(b) Induced ordered subtree T_o' of T



(c) Embedded ordered subtree T_o'' of T



(d) Embedded unordered subtree T_u'' of T



(e) Embedded unordered subtree T_u'' of T

Figure 4.2: Examples of subtrees from a given tree. The initial tree T is illustrated in Figure 4.2a. In Figure 4.2b there is a tree T_o' that is an induced ordered subtree for T and in Figure 4.2c there is a tree T_o'' that is an embedded subtree for T. In Figure 4.2d and in Figure 4.2e are represented another two examples of induced and embedded trees but for the unordered case.

Ordered and unordered trees With respect to the distinction between ordered and unordered trees, we say that a tree is ordered when it is defined an order between the children of a node v. Therefore, if v belongs to an ordered tree, and has N children, there exist a sequence $v_1, ..., v_N$ that represents the order of its children, where $v_i \leq v_{i+1}$, with $0 \leq i \leq N-1$. For example, in Figure 4.1 are illustrated three trees: T_1, T_2, and T_3. If we consider them as unordered trees, they are in fact the same tree, because the ordered between the children of each node does not count. However, we can consider them as three different ordered trees, where the order between the children of the same node can be defined with respect to the disposition from left to right in the visual layout.

Preorder function In addition, we define $preorder(v)$ as the function that returns an index that represents the order of visit of a node v which belongs to a tree T during a pre-order Depth First Search (DFS). For example, if r is the node that represents the root of a tree, $preorder(r) = 1$, and if l is the last right leaf of a tree $preorder(l) = |V|$. For example, in the tree T_1 in Figure 4.1, each node v is labeled with a number representing its pre-order DFS, $preorder(v)$.

Induced and embedded subtree Given the new notations defined above, we can present the *induced* subtrees and *embedded* subtrees defining them as follows[28]:

Definition 8 (Induced subtree)**.** Given a tree T with a set of vertex V and a set of edges E, and a tree T' with a set of vertex V' and a set of edges E', T' is an induced subtree of T if and only if:

1. $V' \subseteq V$ and $E' \subseteq E$
2. The labeling of V' and E' in T is preserved in T'
3. The order among siblings, i.e. the children of the same node, when it exits in T, is preserved in T'

Definition 9 (Embedded subtree)**.** Given a tree T with a set of vertex V and a set of edges E, and a tree T' with a set of vertex V' and a set of edges E', T' is an embedded subtree of T if and only if:

1. $V' \subseteq V$
2. The labeling of V' and E' in T is preserved in T'
3. For every $e \in E'$, where $v_1 = \textit{parent}(e)$ and $v_2 = \textit{children}(e)$, v_1 is an ancestor of v_2 in T.
4. If there exists an order between siblings, then for each $v_1, v_2 \in V$ and $\textit{preorder}(v_1) < \textit{preorder}(v_2)$ in T' if and only if $\textit{preorder}(v_1) < \textit{preorder}(v_2)$ in T

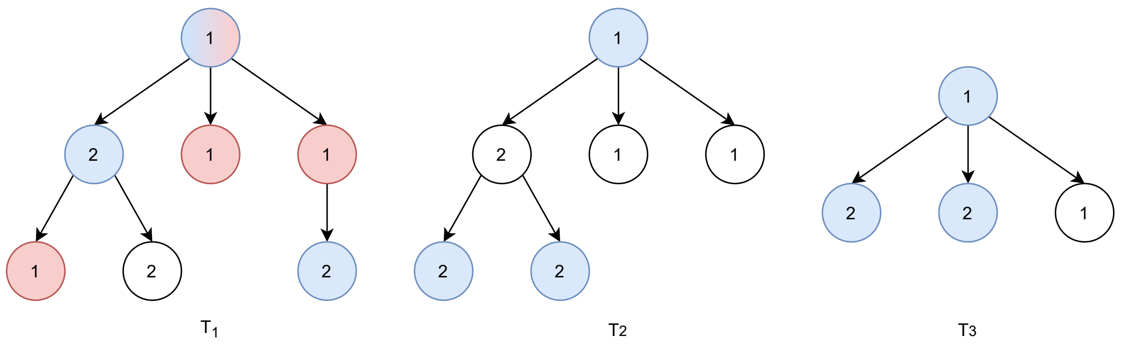
Given the two definitions, it easy to see that if a subtree T' is an induced subtree of T it is also an embedded subtree of T, but the opposite is not always true. Therefore, the definition of the embedded subtree relationship is a generalization of the induced subtree one. An example of the two different types of subtrees, including also the ordered and unordered scenario, is presented in Figure 4.2.

4.2.2 Tree string encodings

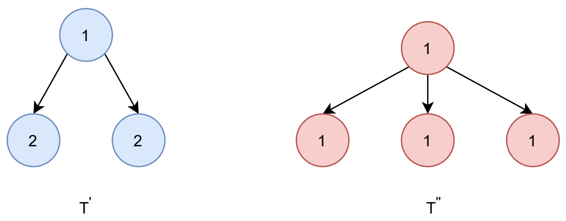
To simplify the process of comparison between trees, various subtree mining algorithms use a canonical string encoding to represent them. The one used in this thesis is the same used by Zaki for TreeMiner[58] and SLEUTH[57] and its described as follows:

Definition 10 (String enconding of a Tree)**.** Given a tree T, its string encoding $\mathcal{T}$ is a sequence of labels generating according to its unique pre-order DFS. In particular, starting from $\mathcal{T}$ equal to the empty string, i.e. $|\mathcal{T}| = 0$, we append each vertex labels of T encountered during the pre-order DFS, and we add a backtrack symbol *\$*, that does not belong to the set of labels, each time we backtrack from a child node to its parent.

For example, the tree T in figure Figure 4.2a has a corresponding $\mathcal{T}$ equal to *1 2 3 \$ 4 \$ \$ 5 \$ 6 7 \$ \$*. In our implementation we use the label *-1* to represent the backtrack symbol \$.



(a) A tree dataset composed by three trees, T_1, T_2, and T_3.



(b) Two possible embedded subtrees T' and T''

Figure 4.3: Example of subtree mining for a dataset $\mathcal{D}$ composed of three trees,T_1, T_2, and T_3 represented in (a). The two embedded subtrees considered are T' and T'', graphically presented in (b). The node colors in the dataset are chosen accordingly to a possible match with the two subtrees. It is worth to notice that T' has multiple occurrences in T_1 and T_2 but to have $d_{T_1}(T') = 1$ or$d_{T_2}(T') = 1$ there must be just an occurrence (or more) of T'.

4.2.3 The subtree mining problem

Before presenting the algorithms used to investigate frequent subtrees in a tree ensemble, we introduce the general subtree mining problem. Given a tree T and a subtree T', we define $\delta_T(T')$ as the number of occurrences of T' in T. It is worth to notice that we define the frequent subtree problem using a general subtree T', it could be either an induced subtree, or an embedded subtree, or another type of subtree.
Then, we define $d_T(T') = \mathbb{1}\{\delta_T(T') > 0\}$, which indicates the presence or absence of T' in T.
Furthermore, let a dataset of trees $\mathcal{D}$, composed by N trees, indicated with T_i, with $1 \leq i \leq N$, we define the support of a subtree T' over a dataset $\mathcal{D}$ of trees as $\sigma_{\mathcal{D}}(T') = \sum_{i=1}^{N} d_{T_i}(T')$, therefore $0 \leq \sigma_{\mathcal{D}}(T') \leq N$.
Given the definitions above, a subtree T' is called frequent in a tree dataset $\mathcal{D}$, if its support is greater of a minim support threshold $minsup$, $\sigma_{\mathcal{D}}(T') \geq minsup$.
For example, in Figure 4.3 the two embedded subtrees considered, T' and T'', have respectively $\sigma_{\mathcal{D}}(T') = 3$ and $\sigma_{\mathcal{D}}(T'') = 1$. Assuming $minsup = 2$, only T' would be a frequent embedded subtree for the dataset $\mathcal{D}$.

4.2.4 Subtree mining algorithms

In this section we review the main characteristics of the algorithms that have been adopted to mining frequent subtrees from a tree ensemble, namely SLEUTH and CMTreeMiner[56].

SLEUTH SLEUTH, anagram of **L**isting "**H**idden" or **E**mbedded **U**nordered **S**ub**T**rees, is a frequent subtree mining algorithm for unordered and embedded subtrees in a dataset of trees proposed by Zaki in 2005[57] as an extension of TreeMiner[58]. The algorithms it is based on a special vertical representation of the dataset called *scope-list*. In the scope-list representation, each label l, from the set of all the labels present in the dataset of trees, is associated with a list of pair (t, s) in which t is a tree ID where l occurs and s is its scope, i.e. the right-most label of the leaf of the tree having as root l. If a label l is present more than once in a tree T_k, multiple pairs of the type (k, s) are added to its scope list. Using this vertical representation of the dataset, the algorithm starts the frequent subtree mining generating all the possible frequent subtrees from a small subset of frequent labels, following the fact that a subtree of k nodes is frequent only if its *prefix* of $k - 1$ nodes is frequent. The candidate generation is then made using a procedure called *scope-list join*, for fast computing new candidates that are also frequent. The scope-list join has been proved to correctly generate all possible **embedded** or **induced**, **unordered** and frequent subtrees.

In some case, in this paper we have also used TreeMiner, that deals only with **embedded** and **induced**, **ordered** subtrees, and it is based on the same technique as SLEUTH, exploiting a scope-list representation to efficiently generate all possible candidates of frequent subtrees.

CMTreeMiner Given the fact that the number of frequent subtrees grows exponentially with the size of the trees, Chi et al. proposed an algorithm to mining only closed and maximal subtrees, CMTreeMiner[55], to reduce the number of trees mined but maintaining most of the relevant information of all the no maximal neither closed subtrees. Specifically, a tree T' is *maximal* if does not exists any proper supertree of T' that is frequent, and a tree T' is *closed* if, given its support $\delta_T(T')$, does not exists any proper subtrees of T' that has the same $\delta_T(T')$. It is easy to show that the number of maximal subtrees is less than the number of closed subtrees, and the number of closed subtrees is less than the number of frequent trees. Formally, $\mathcal{M}_\mathcal{D} \subseteq \mathcal{C}_\mathcal{D} \subseteq \mathcal{F}_\mathcal{D}$, where $\mathcal{M}_\mathcal{D}$ is the set of maximal subtrees, $\mathcal{C}_\mathcal{D}$ is the set of closed subtrees retrieved, $\mathcal{F}_\mathcal{D}$ is the number of frequent trees, and D is the dataset of interest.

The algorithm uses an enumeration Directed and Acyclic Graph (DAG) to compute all the possible maximal and closed subtrees. The enumeration DAG is a graph subdivided into k levels, where, in each level, there are nodes that represent subtrees of the same size. For example, at level 1 the nodes represent all the subtrees composed only by one node, and at level k the nodes represent subtrees composed of k nodes. An edge is present from a node at level k to level $k+1$ if the node at level k is a subtree of the one at level $k + 1$. The mining of the frequent subtrees is made by growing the enumeration DAG (a procedure called CM-Grow) from level one until no further levels can contain frequent subtrees. CMTreeMiner can mine **induced** subtrees from **ordered** and **unordered** trees.

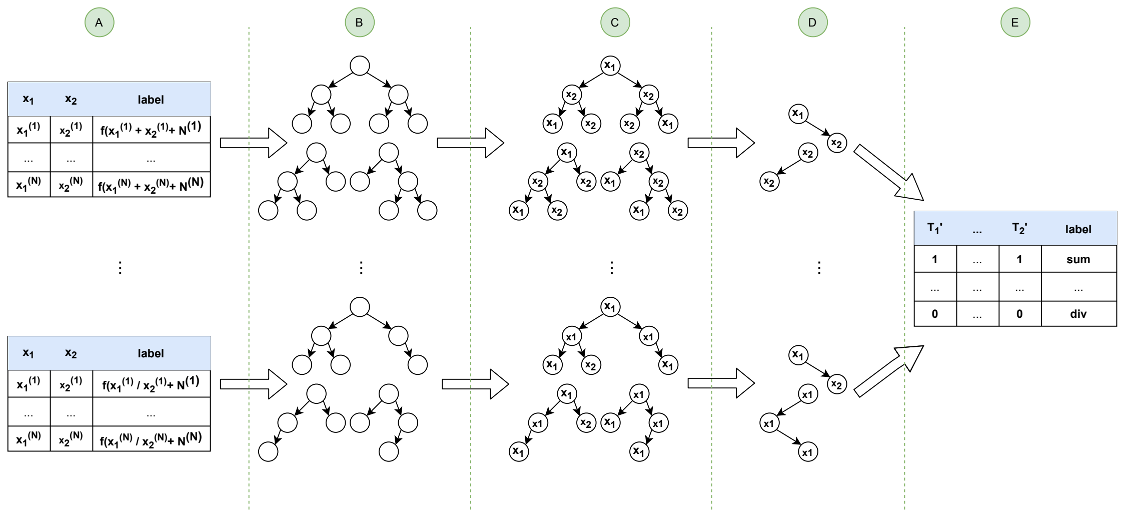


Figure 4.4: A graphic summary of the framework proposed in this section. Each phase is indicated with a circle labeled with a letter from A to E. In phase A there is the generation of all the synthetic datasets representing a specific feature relation. In phase B an ensemble is trained over each dataset. In phase C we transform the ensemble in a tree dataset using the *labeling* function. In phase D we mine all the frequent subtrees from each tree datasets. Finally, in phase E, we create a dataset that we can use to classify the type of interaction from a new ensemble.

4.3 From frequent subtree mining to feature interaction classification

First of all, we present our proposed framework to identify specific interaction between features in a tree ensemble mining frequent subtrees. In this process we can specify 5 different phases, defined as follow:

A The generation of multiple synthetic datasets from a known set of functions under investigation.

B The creation of multiple ensembles, fitted over the dataset created in the previous phase, simulating the models to be explained.

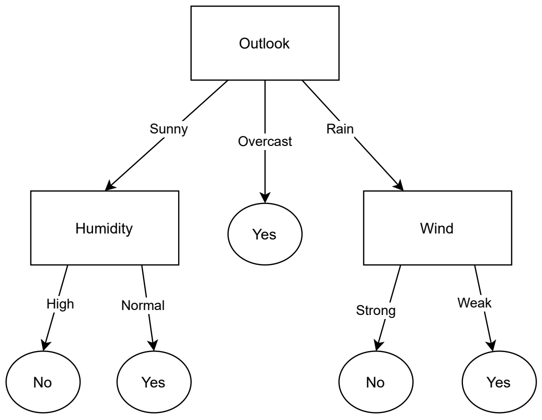
C The transformation from ensembles to tree datasets, where is possible to mine frequent subtrees.

D The mining process of all frequent subtrees from the tree datasets.

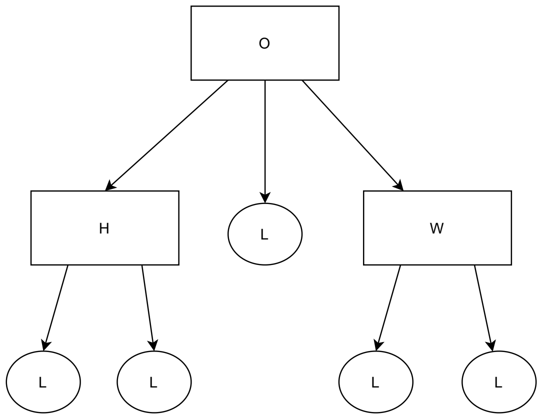
E The creation of a classifier of features interaction, i.e. a classifier used to detect the type of interaction between features from a new unseen ensemble.

The 5 phases introduced above are graphically presented in Figure 4.4 and are describe in detail in following paragraphs.

Generation of synthetic datasets Since we are initially interested only on the interaction between a pair of features we can simplify the problem on a very specific case.



(a) An example of the initial decision tree T.



(b) A transformation of the decision tree, created through the *labeling* function, suitable to mine all its frequent subtrees.

Figure 4.5: Example of transformation from the decision tree presented in section 1.3, that is proposed again in Figure 4.5a, to a labeled tree where each label of a node is related to the splitting feature of the original decision tree Figure 4.5b. In this case the label set $\mathcal{L}$ is equal to $\{O, H, W, L\}$, and $\mathcal{L}_{feat} = \{O, H, W\}$ and $l_l = L$.

Let the real function behind the data used to create an ensemble depending only from two features, $f(\mathbf{x}) = f(\mathbf{x_1}, \mathbf{x_2})$. In this case, the dataset D_N that define the learning problem can be seen as a random sample of $\mathcal{X}$, to which we associate a label that correspond to the function $f(\mathbf{x_1}, \mathbf{x_2})$. Thus, the dataset is equal to $D_N = \{(\mathbf{x}^{(i)}, f(\mathbf{x}^{(i)}) + \mathcal{N}^{(i)})\}_{i=1}^{N}$, where $\mathcal{N}^{(i)}$ is the noise added to the labeling function. In the real world, a source of noise can be for example a low accuracy of a measuring device, or a human error, among others.
Given D_N, we create a tree ensemble with a learning function $L(D_N)$ that give us an approximation of $f(\mathbf{x})$, $L(D_N) = \hat{f}(\mathbf{x})$. We highlight that in all the tree ensemble methods described in this thesis (see section 1.4) $\hat{f}(\mathbf{x})$ is computed by combining a set of K trees, creating a set $\mathcal{E} = \{T_i\}_{i=1}^{K}$.

From a decision tree to a labeled tree The set of all the decision trees that compose the ensemble $\mathcal{E}$ can be easily seen as a dataset of labeled trees, as long as we label all the inner nodes of each decision tree T_i with a label $l_j \in \mathcal{L}$ where j corresponds to the feature used in each split, creating a set of labels $\mathcal{L}_{feat} = \{l_j\}_{j=1}^{M}$, where M is the number of feature used in the decision tree. If we create this correspondence between inner node and labels l_j, the only nodes that remain uncovered are the leaves, that we can label with $l_l \notin \mathcal{L}_{feat}$. We denote this procedure as $labeling(\mathcal{E})$, that produces a dataset of trees from an ensemble $\mathcal{E}$, $labeling(\mathcal{E}) = D_{\mathcal{E}}$. An example of the *labeling* procedure for a single tree T is presented in Figure 4.5.

Mining all the frequent subtrees After the creation of $D_{\mathcal{E}}$ from the initial ensemble of trees $\mathcal{E}$ it is possible to apply a mining algorithm to mine all the frequent subtrees in $D_{\mathcal{E}}$. After this operation, we can create a set of subtrees $\mathrm{T}_{D_{\mathcal{E}}} = \{T'_1, T'_2, ..., T'_k\}$ that represents the set of all frequent subtrees mined from $D_{\mathcal{E}}$, and where for each T'_i we know its support $\sigma_{D_{\mathcal{E}}}(T'_i)$ is greater than a specified

minsup because it has been mined as frequent.
Since the transformation from an ensemble $\mathcal{E}$ to a dataset of tree $D_{\mathcal{E}}$ is easily created, sometimes we use only $\mathcal{E}$ to refer to $D_{\mathcal{E}}$ to not burden the notation and when the connection can be clearly made.
Therefore, given a set composed by dataset of trees $\mathrm{E} = \{D_{\mathcal{E}_j}\}_{j=1}^{N}$ that has been created from N "transformed" ensembles (transformed according to the *labeling* function) we can associate to each ensemble $\mathcal{E}_j$ a set $\mathrm{T}_{\mathcal{E}_j}$ that represents all the frequent subtrees in the specific $\mathcal{E}_j$.

Feature interaction analysis as a classification problem We then finally create a new learning problem where the feature space $\mathcal{X}$ is defined as the union of all the sets of frequent trees $\mathrm{T}_{\mathcal{E}_j}$, i.e. $\mathcal{X} = \bigcup_{j=1}^{N} \mathrm{T}_{D_{\mathcal{E}_j}}$, and the label space $\mathcal{Y}$ the set of all the labels that represent an interaction between two features, such as *sum* for the summation between two features, *prod* for the product between two features, etc.
Thus, the cardinality of the labels $|\mathcal{Y}|$ is equal to the number of function under investigation. For example, if we are considering only the summation (*sum*) and the division (*div*) between two features, $\mathcal{Y} = \{sum, div\}$ and $|\mathcal{Y}| = 2$. This is also consistent with the fact that in RQ 1 we expressed our intention to have control over the dimension of E, the set containing arbitrary complex closed-form expressions, because we can choose how many function include in our analysis.
Furthermore, each sample $\mathcal{E}_j$, i.e. each ensemble, in this classification problem is represented in the n-dimensional feature space $\mathcal{X}$, where n is the number of distinct frequent subtrees used to represent each $\mathcal{E}_j$. In other words, each dimension i, $1 \leq i \leq n$ is associated with a tree T_i, and each vector $\mathbf{x}^{(j)} \in \mathcal{X}$ represents the ensemble $\mathcal{E}_j$.
To represent each ensemble $\mathcal{E}_j$ we initially propose a simple strategy: each vector $\mathbf{x}^{(j)}$ used to represent $\mathcal{E}_j$, contains a 1 for the feature $\mathbf{x}_i^{(j)}$ if the subtree T_i is frequent for $\mathcal{E}_j$ and a 0 when it is not frequent. Therefore, $\mathcal{X}$ is formed by binary features, that represents the presence or absence of a particular frequent subtree in an ensemble. If we want to take into consideration also the frequency of a subtree, and not only its presence or absence, we can replace the ones in the features space with the actual support of each subtree in $\mathcal{E}_j$.
An example of this classification problem is presented in the next section where we investigate if this is a sensible way to classify simple feature interactions inside a tree ensemble.

4.4 Research question 3: Find the relationships between features through pattern discovery

In this section, we illustrate the problem proposed in the section 4.3 and we propose a solution for RQ 3 with an exploratory analysis over a synthetic dataset of ensembles.
We first define how we have created these ensembles to explore our research question, and then, we present a basic exploratory data analysis made mining only the maximal ordered frequent subtrees with CMTreeMiner.
The analysis has followed the 5 phases described in section 4.3, having as a goal the definition of the relation, or not, between the frequent subtrees mined from an

ensemble and its correspondent function learned.

Synthetic datasets definition To explore the feasibility of this approach, we have created 6000 datasets of the type $D_N = \{(\mathbf{x}^{(i)}, f(\mathbf{x}^{(i)}) + \mathcal{N}^{(i)})\}_{i=1}^{N}$ where the function $f(\mathbf{x})$ it has been chosen between three basic algebraic operations: summation, product and division, i.e. we have $f_{sum}(\mathbf{x}) = \mathbf{x}_1 + \mathbf{x}_2 + \mathcal{N}$, $f_{prod}(\mathbf{x}) = \mathbf{x}_1 \cdot \mathbf{x}_2 + \mathcal{N}$, $f_{div}(\mathbf{x}) = \mathbf{x}_1 / \mathbf{x}_2 + \mathcal{N}$. In particular, for each function we have drawn 10000 samples, thus $N = 10000$, from two different distributions to assess if the feature distribution can impact the similarity between the same type of function $f(\mathbf{x})$ used. The two distribution selected are a uniform distribution between -100 and 100 — $\mathcal{U}(-100, 100)$ — and a normal distribution with mean 0 and standard deviation equal to 100, $\mathcal{N}(0, 100)$. The noise $\mathcal{N}$ has been drawn from a normal distribution $\mathcal{N}(0, 5)$ to add a small perturbation on the label that is normal to have in a real case scenario. Therefore, in total, we have generated 6000 datasets, 2000 for each type of function.

Ensemble learning and preprocessing Over the datasets described in the previous paragraph, 6000 tree ensembles have been learned via LGBM using GBDT. Specifically, all the tree ensemble has been learned without hyperparameters tuning and using all the default settings provided by the library. Thus, all the forest investigated are composed of 100 trees, with a number of leaves equal to 31, and without any limitations for the max depth of the trees. After the creation of all the ensembles, they have been transformed in tree datasets using the *labeling* function described in section 4.3.

Mining maximal subtrees The last step for the final dataset creation is the mining of all the maximal subtrees from all the ensembles with a $minsup = 75\%$, i.e. a subtree is frequent if it is present in the 75% of the trees composing the ensemble.
After having mined all the frequent subtrees we can create tuples $(\mathcal{E}_j, \mathrm{T}_{D_{\mathcal{E}_j}})$, where $\mathcal{E}_j$ is the j-th ensemble and $\mathrm{T}_{D_{\mathcal{E}_j}}$ is the set of frequent subtree retrieved from the specific ensemble. As described in section 4.3, we then create a set of all the distinct subtrees retrieved that will represent the feature space for the classification problem, where each ensemble is associated to a label that represents the type of its features interaction.
In this case $\mathcal{X} = [0, 1]^{246}$, since we have retrieved 246 different subtrees from all the ensembles under investigation. In average each ensemble has been associated with almost 10 different frequent subtrees, with a maximum of 19 and a minimum of 4. On the other hand, $\mathcal{Y} = \{sum, prod, div\}$ where *sum* has been associated with all the ensembles that have be trained over a dataset created from $f_{sum}(\mathbf{x})$, *prod* with $f_{prod}(\mathbf{x})$, and *div* with $f_{div}(\mathbf{x})$.
A table to describe the new classification problem is presented in Table 4.1 to further illustrate how the new problem appears.

Exploratory analysis We start the investigation over the dataset created with a Principal Component Analysis (PCA)[54] to reduce the feature space from 246 features to 2 or 3 principal components that represent a combination of the initial features and they try to explain as much of the variance of the initial space. The purpose of this space transformation is to visualize all the samples created and see

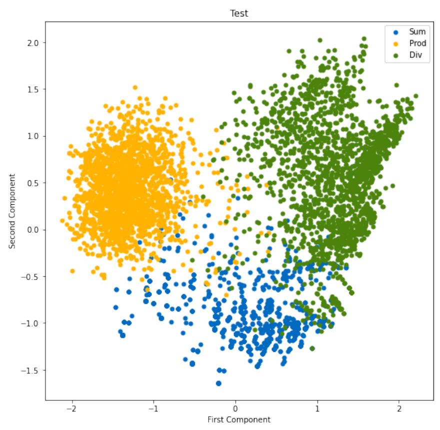
Feature	Description
1 1 -1 2 -1	First String encoding of a possible subtree where *1* represents $\mathbf{x}_1$ and *2* represents $\mathbf{x}_2$. It is set to 1 (or true) if it is present in the ensemble, 0 (or false) otherwise.
⋮	⋮
1 2 2 -1 1 -1 -1	Last String encoding of a possible subtree. In total there are 246 possible frequent subtrees.
label	A label indicating the type of relationship between the features, in this case we have considered only three basic relationships: *sum* for the summation of the features, *prod* for the product of the two features, and *div* if we are dealing with the division between the two.

Table 4.1: Table summarizing the features used in the last phase of the analysis: the classification of the type of interaction. The dataset is composed by 6000 samples (i.e. 6000 tree ensembles), represented with 246 features that describe the presence or absence of a particular frequent subtrees in the forest. Each sample is associated with a label belonging to the set $\mathcal{Y} = \{sum, prod, div\}$, that describes the type of underline interaction present in the forest. There are 2000 sample labeled with *sum*, 2000 samples labeled with *prod*, and 2000 samples labeled with *div*.

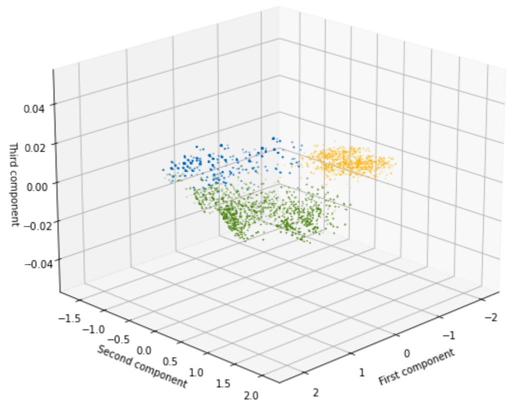
if there are some structures that can suggest a correlation between the frequent subtrees mined and the functions learned. Even though with a 2D PCA we are able to explain only the 30% ca. of the initial variance, and almost the 36% ca. with a 3D PCA, from Figure 4.6 it is easy to see that the samples can be easily subdivided into three big clusters that correspond to the three different functions under analysis, confirming our initial hypothesis. To confirm the easy subdivision of the three groups, we have created a simple decision tree classifier adopting as a splitting criterion the information gain and with a maximum depth equal to two, i.e. a very shallow tree as shown in Figure 4.7. For this learning problem, we have subdivided the dataset into two: 70% of the dataset has been used as the training set and 30% as the test set.

From the decision tree in Figure 4.7 we can see that there are three main subtrees that are discriminative according to the decision tree, namely: *2 1 -1 1*, *2 1 1 2 -1 -1 -1*, and *1 2 2 2 -1 -1 -1*. In fact, as we can see from the three subtrees atlases in Figure 4.9, a substructure of the type *2 1 -1 1* is the most frequent in the *sum* class, and it is not present in the other atlas, meaning that there are at least other five subtrees that are more frequent than *2 1 -1 1*, and this gives us a signal that we are dealing with a *sum* relationship between the features. The same rational can be applied to *2 1 1 2 -1 -1 -1* that is the most frequent for the *div* class. Also *1 2 2 2 -1 -1 -1* is a very frequent pattern on average for *div*, even though not in the atlas to reduce the space of the figure.

In addition a confusion matrix created from the evaluation of the test set is available at Table 4.2, showing an high accuracy to discriminate between almost all the classes. We only highlight a lower accuracy to predict the *div* class, however it can be easily improved increasing the depth to of the decision tree. In fact, only bringing the decision tree depth to three can give a 92% of accuracy in predicting



(a) 2D PCA scatter plot



(b) 3D PCA scatter plot

Figure 4.6: Results of the PCA over the dataset that represent all the ensembles via their frequent subtree patterns. In Figure 4.6a is presented a plot with the first 2 components and in Figure 4.6b a 3D plot with the first three components. In blue are represented the ensembles learned from a *sum* function, in orange the ones learned from a *product* function, and in green the ones learned from a *division* function.

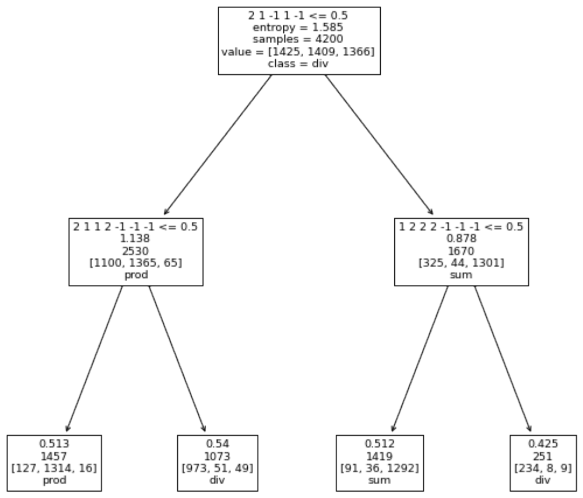


Figure 4.7: The classifier learned to distinguish between the three classes *sum*, *prod* and *div*. In each node there are five information available: the splitting criterion, the value of the entropy, the number of samples, the subdivision of the samples in each class, and the predicted class (i.e. the class that is more present in the splitting associated with the node).

True label \ Predicted label	*div*	*prod*	*sum*
div	0.81	0.11	0.08
prod	0.056	0.93	0.019
sum	0.035	0.021	0.94

Table 4.2: The confusion matrix of the prediction made by the classifier created and presented in Figure 4.7. The values have been normalized by rows, and the true positive rates for each category have been highlighted in different colors.

also the *div* class; here we present the results only for a decision tree with depth equal to two because it is more explainable having only three pattern involved instead of seven.
To conclude the exploratory analysis, we have investigated the information gain (or mutual information) from each pattern to analyze how much discriminative they are for the problem[10]. Thus we present in Figure 4.8 a bar chart displaying the first 20 subtrees with the highest values of information gain on the horizontal axis (sorted by the value of information gain) and the portion of the sample where they are present in the vertical axis, grouped by their label of membership. We recall, as described in section 1.1, that the information gain $I(\mathrm{X};\mathrm{Y})$ between two r.v. X and Y is defined as:

$$I(\mathrm{X};\mathrm{Y}) = \sum_{y\in\mathcal{Y}}\sum_{x\in\mathcal{X}} p_{(X,Y)}(x,y)\log\left(\frac{p(X,Y)(x,y)}{p_X(x)p_Y(y)}\right)$$

in this case X is the r.v. representing the feature space and Y represents the label space, $p_{(X,Y)}(x,y)$ has been estimated using the contingency table between X and Y, and $p_X(x)$ and $p_Y(y)$ have been estimated computing the marginal distribution of the features and the labels.
It is worth noticing that among the first 20 subtrees with the highest values of information gain that are a lot of isomorphic subtrees, such as *2 1 -1 1 -1* and *1 2 -1 2 -1*, or *1 1 1 2 -1 -1 -1* and *2 2 2 1 -1 -1 -1*. On the other hand, it is also not true for all the subtrees, for example for the pattern with the third-highest information gain, namely *2 1 1 2 -1 -1 -1*, its isomorphism with the label inverted, i.e. *1 2 2 1 -1 -1 -1*, it is not present among the first 20 highest informative patterns. Thus, we continued the analysis considering all the patterns and without combining the isomorphisms of the same pattern in a single feature. Another point to highlight is that we have decided to mine all the frequent maximal **ordered** subtrees, and thus we have considered different the pattern *1 1 -1 2 -1* from *1 2 -1 1 -1* even though they represent the same unordered tree. This decision has been made facing a much higher computation time to mine all the frequent maximal **unordered** subtrees with respect to the ordered ones. Given this computational limitation, we have considered to post-process all the subtrees identified and store them with a *canonical* representation, in order to merge all the subtrees differing only for the sibling's nodes order. However, this canonical

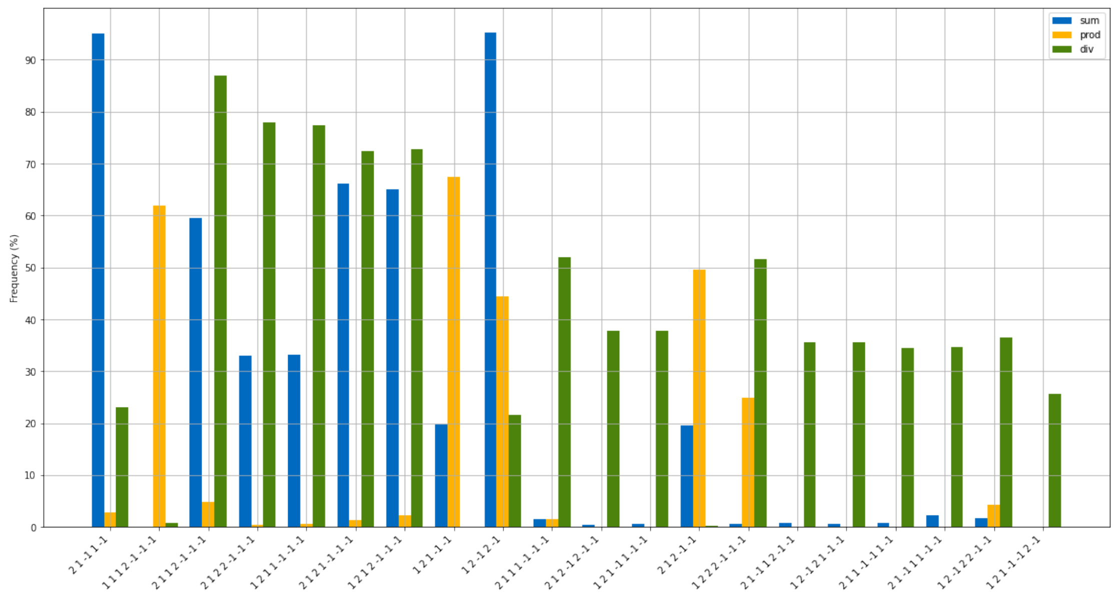


Figure 4.8: Grouped bar chart to visualize the information gain over the initial dataset. In each group three bar the three labels: sum, prod, and div. The height of each bar corresponds to the share of samples of that class in which that subtree is considered frequent. The patterns in the horizontal axis are sorted by information gain.

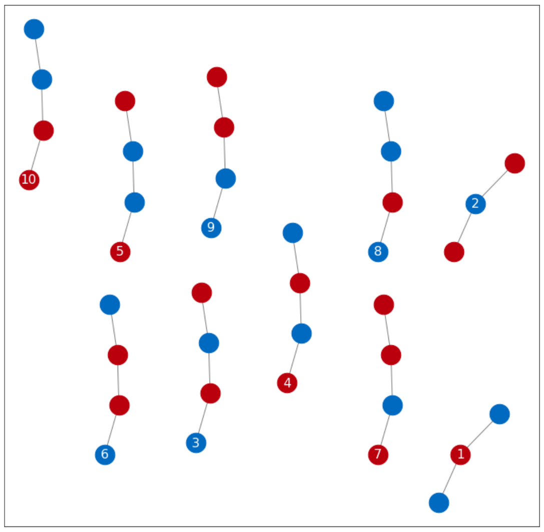
representation of the subtrees had not a significant impact on the classification performance, thus we decided to skip this additional step and leave it for possible further investigations.

4.5 Research question 2: Forest explanation through frequent subtrees

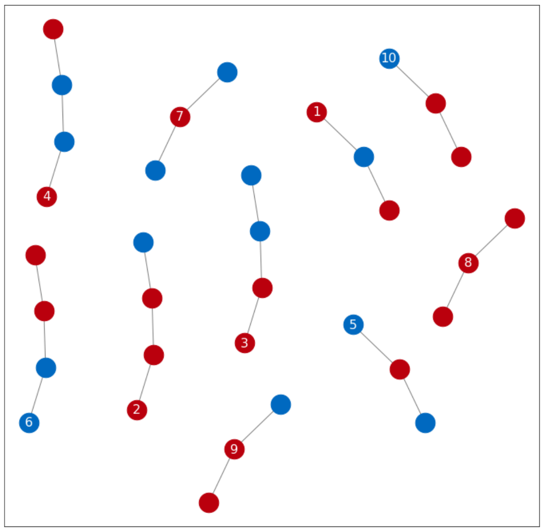
In the previous section, we have tried to answer to RQ 3, analyzing the frequent subtrees created from ensemble trained over dataset drawn from functions of two variables. In this section, instead, we investigate a possible solution for RQ 2. In particular, given the relation found between a function learned by an ensemble and its frequent maximal subtrees, we investigate if a function learned by a tree ensemble $\hat{f}(\mathbf{x})$ that tries to approximate a composition of function in the form $f(\mathbf{x}) = g(h(\mathbf{x}))$ have some frequent subtrees of $h(\mathbf{x})$. If this is the case, we could construct the explanation function $f_e(\mathbf{x})$ by combining the information from each pair of features in an iterative way, retraining the ensemble over an augmented dataset where we add a new feature that represents $h(\mathbf{x})$, and iterate until all features have been considered. To analyze the proposed approach we follow 3 main steps:

1 We repeat steps 1, 2, and 3 of the previous section, adding a new dataset drawn from a *random* function, that represents the absence of relationships between the features.

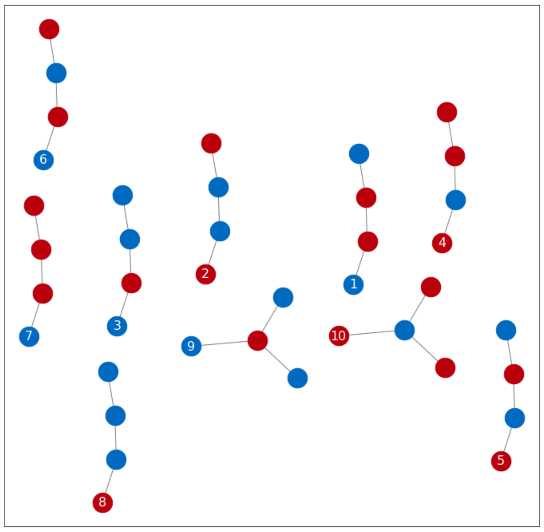
2 We create a test set of ensembles trained over datasets that are drawn from compositions of the functions studied in the previous section.



(a) Most frequent maximal subtrees for the *sum* relationship



(b) Most frequent maximal subtrees for the *prod* relationship



(c) Most frequent maximal for the *div* relationship

Figure 4.9: Three atlases for the corresponding three relationships taken into account. In each atlas, the root of the subtree is indicated with a number $1 \leq k \leq 5$. In particular $k = 1$ represents the most frequent tree, $k = 2$ the second most frequent, and so on.

3 We propose a framework to classify the relationship between two features in the test ensembles mining all the frequent **embedded** subtrees.

4.5.1 Synthetic datasets definitions

Synthetic datasets from random function To add an additional check to the discrimination power of the frequent subtrees mined from tree ensembles that have been trained over dataset formed by f_{sum}, f_{prod}, and f_{div}, we add 2000 samples of $f_{rnd} = \mathcal{N}(0, 1000)$, and we labeled the tree ensembles trained over these dataset with *rnd*. We add f_{rnd} to our set of datasets to exclude that random patterns, i.e. patterns that do not depend on the relations between the features, could impact the accuracy of the classifier created to label each feature relation. To do that, we do the same procedure described in the previous section to create the decision tree to classify the three relations under analysis, but instead of using a dataset where only the presence or absence of a subtree is stored, we store also the value of its support $\sigma_D(T')$, where D is the tree dataset derived from the tree ensemble. We highlight that since we are mining all the subtrees with $minsup = 75$, the feature $\mathbf{x}_i$ that represent a subtree i, can be or equal to 0, or greater than the $minsup$, i.e. $\mathbf{x}_i = 0 \vee \mathbf{x}_i \geq minsup$. A 10-fold cross validation of a tree ensemble classifier C trained over the dataset created by the 8000 tree ensembles with the four labels (*sum*, *prod*, *div*, and *rnd*) showed an average accuracy of almost 99%.

Synthetic test datasets formed by three features To check if frequent subtrees are linked with the relation behind every pair of features, we create 9 new synthetic datasets, that are drawn from composed functions defined as follows:

$$
\begin{aligned}
f_{sum,sum}(\mathbf{x}) &= \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 + \mathcal{N} \\
f_{sum,prod}(\mathbf{x}) &= (\mathbf{x}_1 + \mathbf{x}_2) \cdot \mathbf{x}_3 + \mathcal{N} \\
f_{sum,div}(\mathbf{x}) &= \frac{(\mathbf{x}_1 + \mathbf{x}_2)}{\mathbf{x}_3} + \mathcal{N} \\
f_{prod,sum}(\mathbf{x}) &= \mathbf{x}_1 \cdot \mathbf{x}_2 + \mathbf{x}_3 + \mathcal{N} \\
f_{prod,prod}(\mathbf{x}) &= \mathbf{x}_1 \cdot \mathbf{x}_2 \cdot \mathbf{x}_3 + \mathcal{N} \\
f_{prod,div}(\mathbf{x}) &= \frac{\mathbf{x}_1 \cdot \mathbf{x}_2}{\mathbf{x}_3} + \mathcal{N} \\
f_{div,sum}(\mathbf{x}) &= \frac{\mathbf{x}_1}{\mathbf{x}_2} + \mathbf{x}_3 + \mathcal{N} \\
f_{div,prod}(\mathbf{x}) &= \frac{\mathbf{x}_1}{\mathbf{x}_2}\mathbf{x}_3 + \mathcal{N} \\
f_{div,div}(\mathbf{x}) &= \frac{\mathbf{x}_1}{\mathbf{x}_2}\frac{1}{\mathbf{x}_3} + \mathcal{N}
\end{aligned}
$$

Thus, we create 100 datasets for each new function defined, i.e. 900 datasets in total, a for each dataset we train a tree ensemble via LGBM with the same hyperparameters used for the functions of only two features, creating in total 900 new ensembles. To not add unnecessary complexity to the procedure, we drawn all the samples $\mathbf{x}$ from a uniform distribution $\mathcal{U}(-100, 100)$. These new ensembles have been created to test if we can classify the type of relation between $\mathbf{x}_1$ and $\mathbf{x}_2$ given the frequent subtrees retrieved from each ensemble.

4.5.2 Proposed procedure

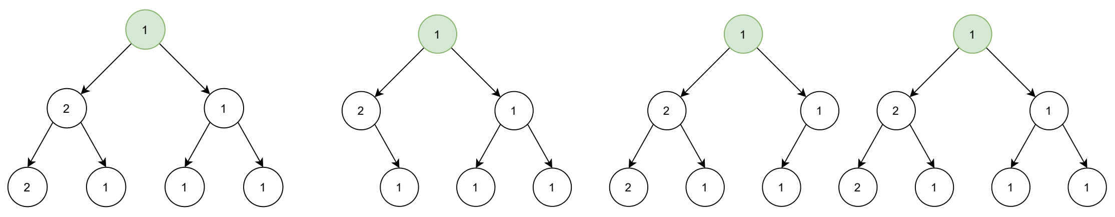
To classify the relation between $\mathbf{x}_1$ and $\mathbf{x}_2$, labeled respectively l_1 and l_2 in the string representation, we propose a new procedure summarized in Algorithm 4. It is divided into three main parts: the mining of all the frequent subtrees, the filtering of only relevant subtrees for the classification problem, and their transformation into the feature space of a given classifier. The actual implementations of all these steps are described in the following paragraphs.

Algorithm 4 Miner Explainer Procedure

```
1: procedure MinerExplainer($E, M, C, (l_1, l_2)$)
2:     ▷ Ensemble $E$, frequent subtree miner $M$, a relation classifier $C$, and the
       couple of feature of interest $(l_1, l_2)$.
3:     $FS \leftarrow M(E)$                          ▷ Mine all the frequent subtrees of $E$ with $M$
4:     $FS_{(l_1,l_2)} \leftarrow filter(FS)$                ▷ Filter only the relevant pattern
5:     $\mathbf{x} \leftarrow transform(FS_{(l_1,l_2)})$      ▷ represent $FS_{(l_1,l_2)}$ in the space $\mathcal{X}$ of $C$
6:     return $C(\mathbf{x})$                                        ▷ Classify $\mathbf{x}$
7: end procedure
```

Mining algorithm First of all, we retrieve all the frequent subtrees with a function $M(E)$ where E is the ensemble under investigation, represented using the *labeling* function. In our implementation, we retrieve not only the maximal induced subtrees with CMTreeMiner as done in the previous analysis, but we mine all the frequent **embedded** subtrees with SLEUTH. We moved from mining only the maximal subtrees to mining all the frequent embedded subtrees because the number of maximal subtrees with $minsup = 75\%$ in the ensembles described above is nearly 0. That means that, for example, given a forest trained over a dataset generated from $f_{sum,sum}$, if we mine all the maximal induced frequent subtrees with $minsup = 75\%$ we can find only subtrees containing no more than one or two nodes.
In this case, using a smaller *minsup* to augment the number of maximal subtrees mined was also taken into account, however, it was not an optimal solution, because, with the algorithm used, we could mine only **induced** subtrees. The problem of mining only induced subtrees is that, in an ensemble with multiple features involved, the frequent subtrees mined include, for the majority, not only two labels, but all the labels used by the forest.
Therefore, we "relaxed" the type of mining and we mine all the frequent subtrees, and not only the maximal ones, to try to retrieve more trees that could be informative. Besides, we moved from mining only induced subtrees to embedded subtrees because we notice that using the latter we were missing a lot of possible discriminative subtrees that were *embedded* in the tree dataset and that were similar to the patterns found over the analysis described in the previous section.

Filtering non relevant subtrees Then, we highlight that the frequent subtrees mined from an ensemble trained over a dataset containing three features can include patterns formed by all three different features, or features different from the pair under investigation (l_1, l_2). Thus, a filter procedure needs to be implemented to select only the meaningful subtrees. In our implementation, we have discarded



(a) A subtree T'. (b) A set of tree T that are subtree of T'.

Figure 4.10: Figure that visually represent the heuristic used to compute the number of frequent subtrees in a tree dataset. Assuming that we want to know if a subtree T', as the one presented in Figure 4.10a, is frequent in a tree dataset, we firstly find the set T of all the other frequent subtrees, as the ones presented in Figure 4.10b, that are in turn subtrees of T' radicated on the same root and have more than $\lfloor n \cdot k \rfloor$ nodes, where n is the number of nodes of T' and $0 < k < 1$ is a predefined constant. In our implementation $k = 0.75$, thus all the subtrees in T, for this example, must have at least 6 nodes. Then we compute the frequency of T' as the maximum frequency of the trees in T.

all the subtrees that contain a different label from the two under investigation: l_1 and l_2. Which means that let T' a frequent subtree, $\mathcal{T}'$ its associated string representation and L the set of different label present in $\mathcal{T}'$, we consider only patterns where $L \subseteq \{l_1, l_2, \$\}$, i.e. where the only labels in its string representation are the two under investigation or the backtrack symbol. In addition, we also discarded all the subtrees composed by at most two nodes, i.e. 1, 2, *1 2*, etc. because they result frequent in almost all the ensembles and thus they arguably can be considered informative. In addition, they negatively interfere with the heuristic presented in the next chapter.

Representation as a point in the feature space After having filtered out the no relevant frequent subtrees, we have to transform the frequent subtrees found in a vector $\mathbf{x}$ that can be classified from the classifier C, i.e. we want that $\mathbf{x} \in \mathcal{X}$.
First of all, it is necessary to solve a label mapping problem, that is we need to map the label under investigation l_1, and l_2, with the label used by our classifier, that in our implementation are *1* and *2*. To do that, we notice that are only two possible mappings: if we map l_1 into *1*, we need to map consequently l_2 into *2*, and, on the other hand, if we map l_1 into *2*, we must map l_2 into *1*. Since we do not know what is the best mapping in advance, in our procedure we try both and we take the one that achieves the highest probability in the classification process. Thus we iterate the following process twice, one for each mapping.
After having decided the initial mapping of the features, we need to find the actual value for each feature of the $\mathbf{x}$ for our classification problem. One simple method to that, it could be to assign to the feature $\mathbf{x}_{T'}$ of $\mathbf{x}$, associated with the subtree T', a value equal to the support $\sigma_D(T')$ retrieved from the $M(E)$ step. However, the results with this simple procedure were not satisfactory and we tried a different approach based on the assumption that if we have found a subtree with $k \cdot n$ nodes, with $0 < k < 1$, it might be only the first part of a bigger subtree of n nodes. Thus, we propose the following heuristic: let $\mathbf{x}_{T'}$ being the feature associated with the frequent subtree T' for $\mathbf{x}$, we retrieve all the subtrees T'' in the set of all frequent subtrees for the relationship under investigation with the following characteristics:

- T'' must be an induced ordered subtree of T', having the same root as T'.

	Label predicted			
Generating function	*sum*	*prod*	*div*	*rnd*
$\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 + \mathcal{N}$	100	0	0	0
$(\mathbf{x}_1 + \mathbf{x}_2) \cdot \mathbf{x}_3 + \mathcal{N}$	59	9	32	0
$(\mathbf{x}_1 + \mathbf{x}_2)/\mathbf{x}_3 + \mathcal{N}$	1	54	45	0
$\mathbf{x}_1 \cdot \mathbf{x}_2 + \mathbf{x}_3 + \mathcal{N}$	0	0	100	0
$\mathbf{x}_1 \cdot \mathbf{x}_2 \cdot \mathbf{x}_3 + \mathcal{N}$	0	61	39	0
$\mathbf{x}_1 \cdot \mathbf{x}_2/\mathbf{x}_3 + \mathcal{N}$	0	14	86	0
$\mathbf{x}_1/\mathbf{x}_2 + \mathbf{x}_3 + \mathcal{N}$	0	22	78	0
$\mathbf{x}_1/\mathbf{x}_2 \cdot \mathbf{x}_3 + \mathcal{N}$	0	76	24	0
$\mathbf{x}_1/\mathbf{x}_2/\mathbf{x}_3 + \mathcal{N}$	0	65	35	0

Table 4.3: Table containing the prediction of the classifier created, each row represented the function from which the data has been generated, and each column represent the predicted label. The true positive are highlighted according to the three different function under investigation *sum*, *prod*, and *div*.

	sum	*prod*	*div*
precision	1.00	0.25	0.31
recall	0.53	0.25	0.45
accuracy	0.41		

Table 4.4: Table with the value of *precision*, and *recall*, for the three classes under investigation: *sum*, *prod*, and *div*. The overall accuracy is presented in the last row.

- If n is the number of nodes in T', T'' must have more than $\lfloor k \cdot n \rfloor$ nodes, where $0 < k < 1$. In our implementation $k = \frac{3}{4}$.

We call the set of the subtrees that respect the proprieties above T. We then compute the value of each feature for $\mathbf{x}$ in $\mathcal{X}$ as the maximum support in the set T, $\mathbf{x}_{T'} = max(\{\sigma_D(T'') : T'' \in \mathrm{T}\})$, where max return the maximum value in a set. For example, given the visual representation of a possible tree T' and an associated set of subtrees T presented in Figure 4.10, if the supports of the three subtrees in T are respectively 75, 77, and 91, the value for the feature $\mathbf{x}_{T'}$ is 91, $\mathbf{x}_{T'} = 91$.

Evaluation To evaluate the aforementioned strategy, we have classified the relation between the features $\mathbf{x}_1$ and $\mathbf{x}_2$ over the ensemble learned from the datasets defined in subsection 4.5.1. The results for each function taken into account are presented in Table 4.3, and in Table 4.4 are presented the associated values of *precision* and *recall* for the three main classes *sum*, *prod*, and *div*, and the value of the overall *accuracy*.

Let TP the true positive predictions, FP the false positive predictions, TN the true negative predictions, and FN the false negative predictions, we recall the the three definitions of *precision*, *recall*, and *accuracy*:

$$\begin{aligned} precision &= \frac{TP}{TP+FP} \\ recall &= \frac{TP}{TP+FN} \\ accuracy &= \frac{TP+TN}{TP+TN+FP+FN} \end{aligned}$$

Given the obtained results, we cannot say that the individuation of the type of relation in the test ensembles between $\mathbf{x}_1$ and $\mathbf{x}_2$ is an easy classification problem as the one presented in section 4.4. We notice from the results obtained that the procedure described has a bias with the *prod* class and a lot of samples are classified as *div*, that is why the recall is low (under 50%) for both *sum* and *product*. However, it is good to notice that with the procedure described there are no relation classified as random (*rnd*) which means that we are able to find if between two features there is something more than a random relation. In addition, we can see that with this classifier we have the maximum precision when the class *sum* is predicted, which means that when we predict an additive relation between two features, in the setting proposed, we are almost sure that the prediction is correct. Finally, we highlight that, even though the outcomes are not completely satisfactory, given the results from *precision*, *recall*, and *accuracy* of the classifier we can say that some frequent subtrees can indeed be an indication of a particular type of relation between the features, and we can do better than a weak learner. However, there is still work to do to discover the best way to reuse the frequent subtrees created from simple functions to find the relations between features when the complexity increases.

4.6 Research question 4: Improve the accuracy through pattern discovery

In order to answer RQ 4, we assume two scenarios: in the first one we estimate a empirical upper-bound, in which we are able to identify correctly for each ensemble the relation between $\mathbf{x}_1$ and $\mathbf{x}_2$; in the second one we use the classification given from the procedure presented in the previous section.
In the first scenario, we assume that for $f_{sum,sum}(\mathbf{x})$ we are able to classify as *sum* the relation between $\mathbf{x}_1$ and $\mathbf{x}_2$, in $f_{prod,sum}(\mathbf{x})$ we are able to classify as *prod* the relation between $\mathbf{x}_1$ and $\mathbf{x}_2$, and so on for each function described in the previous section. After that, we add a new feature to represent this relation in the training dataset. For example, in the case of a training dataset D_t, drawn from $f_{sum,sum}(\mathbf{x})$, we create a new dataset D'_t adding a new feature $\mathbf{x}_4 = \mathbf{x}_1 + \mathbf{x}_2$ to D_t. We then train two forest, one over D_t and the other one over D'_t, and we compare the two accuracies to empirically check if there are any improvements.
In the second scenario, we follow the same steps but using the classification of the relation between $\mathbf{x}_1$ and $\mathbf{x}_2$ given from the procedure presented in the previous chapter.
In this section, we first describe in detail the settings of this performance analysis and we then present and comment the results obtained for the empirical upper-

Generating function	$\mathbf{x}_4$
$\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 + \mathcal{N}$	$\mathbf{x}_1 + \mathbf{x}_2$
$(\mathbf{x}_1 + \mathbf{x}_2) \cdot \mathbf{x}_3 + \mathcal{N}$	$\mathbf{x}_1 + \mathbf{x}_2$
$(\mathbf{x}_1 + \mathbf{x}_2)/\mathbf{x}_3 + \mathcal{N}$	$\mathbf{x}_1 + \mathbf{x}_2$
$\mathbf{x}_1 \cdot \mathbf{x}_2 + \mathbf{x}_3 + \mathcal{N}$	$\mathbf{x}_1 \cdot \mathbf{x}_2$
$\mathbf{x}_1 \cdot \mathbf{x}_2 \cdot \mathbf{x}_3 + \mathcal{N}$	$\mathbf{x}_1 \cdot \mathbf{x}_2$
$\mathbf{x}_1 \cdot \mathbf{x}_2/\mathbf{x}_3 + \mathcal{N}$	$\mathbf{x}_1 \cdot \mathbf{x}_2$
$\mathbf{x}_1/\mathbf{x}_2 + \mathbf{x}_3 + \mathcal{N}$	$\mathbf{x}_1/\mathbf{x}_2$
$\mathbf{x}_1/\mathbf{x}_2 \cdot \mathbf{x}_3 + \mathcal{N}$	$\mathbf{x}_1/\mathbf{x}_2$
$\mathbf{x}_1/\mathbf{x}_2/\mathbf{x}_3 + \mathcal{N}$	$\mathbf{x}_1/\mathbf{x}_2$

Table 4.5: Table summarizing the new values of the fourth feature $\mathbf{x}_4$ computed for each dataset.

bound and for the results obtained with the classification procedure presented in the previous section.

Setup To estimate the impact of the addition of the new features to the training datasets under investigation, we used the same set of datasets drawn from the composed functions described in the previous section. For each ensemble we evaluate the Mean Squared Error (MSE) over a test dataset composed by 1000 samples and labeled according the same function used in the training set. We recall that the MSE of an ensemble E over a test dataset $D_v = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$ is defined as $MSE = \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - \hat{y}^{(i)})$, where $\hat{y}^{(i)}$ is the prediction given from the ensemble E for $\mathbf{x}^{(i)}$. Since we have 100 dataset drawn from each type of function, we have trained 100 tree ensembles for each function, which means 100 ensembles for $f_{sum,sum}(\mathbf{x})$, 100 ensemble for $f_{sum,prod}(\mathbf{x})$, etc. and then we have computed the MSE before adding the new feature and the MSE after the addition of the new feature representing the relation found. In Table 4.5 is presented a table that summarize which are the hypothetical new features for each type of generating function.

Results — empirical upper-bound In order to get an overview of the results obtained we have computed the average $MSE_{old} = \frac{1}{K}\sum_{i=1}^{K} MSE_{fun}^{(i)}$ where $MSE_{fun}^{(i)}$ is the MSE of the i-th ensemble trained over a dataset drawn from the function fun with respect to the test set; in our setting $K = 100$. On the other hand MSE_{new} have the same definition, but it has instead been computed on the ensembles trained over the augmented dataset.
The results of the MSE computation before and after the feature engineering process, where a new feature is added to each training dataset to represents a relation between $\mathbf{x}_1$ and $\mathbf{x}_2$ are presented in Table 4.6. From the variation obtained, we can say that in the majority of the case, adding a new feature to the training dataset according to the procedure described above can greatly improve the performance of the tree ensemble. In fact, we notice an improvement of the performance for 9 types of function, over the 12 under investigation. We also notice we had a significant degradation of the performance only on ensembles trained over datasets drawn by functions that contain two particular types of operations, namely the multiplication, and the division, i.e. $f_{prod,div}$ and $f_{div,prod}$. The strange results obtained for $f_{prod,div}$ and $f_{div,prod}$ are probably due to the fact that the ensemble is

Generating function	MSE_{old}	MSE_{new}	Δ_{MSE} %
$\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 + \mathcal{N}$	$1.73 \cdot 10^1$	$6.43 \cdot 10^0$	63.01
$(\mathbf{x}_1 + \mathbf{x}_2) * \mathbf{x}_3 + \mathcal{N}$	$1.16 \cdot 10^5$	$6.08 \cdot 10^4$	47.68
$(\mathbf{x}_1 + \mathbf{x}_2) / \mathbf{x}_3 + \mathcal{N}$	$3.54 \cdot 10^7$	$3.55 \cdot 10^7$	-0.14
$\mathbf{x}_1 * \mathbf{x}_2 + \mathbf{x}_3 + \mathcal{N}$	$2.81 \cdot 10^4$	$9.70 \cdot 10^3$	65.51
$\mathbf{x}_1 * \mathbf{x}_2 * \mathbf{x}_3 + \mathcal{N}$	$1.19 \cdot 10^9$	$5.49 \cdot 10^8$	53.87
$\mathbf{x}_1 * \mathbf{x}_2 / \mathbf{x}_3 + \mathcal{N}$	$9.67 \cdot 10^7$	$1.63 \cdot 10^8$	-68.83
$\mathbf{x}_1 / \mathbf{x}_2 + \mathbf{x}_3 + \mathcal{N}$	$1.36 \cdot 10^5$	$1.26 \cdot 10^5$	7.69
$\mathbf{x}_1 / \mathbf{x}_2 * \mathbf{x}_3 + \mathcal{N}$	$6.09 \cdot 10^7$	$7.18 \cdot 10^7$	-17.86
$\mathbf{x}_1 / \mathbf{x}_2 / \mathbf{x}_3 + \mathcal{N}$	$1.16 \cdot 10^4$	$1.01 \cdot 10^4$	12.60

Table 4.6: Average MSE of ensembles before adding a new feature in the training dataset that represent the relation between $\mathbf{x}_1$ and $\mathbf{x}_2$ (MSE_{old}), and after (MSE_{new}). The last column $\Delta_{MSE}\%$ represents the variation between the two quantities as $\Delta_{MSE}\% = (MSE_{old} - MSE_{new})/MSE_{old} \cdot 100$, where $-100 \leq \Delta_{MSE}\% \leq 100$, and a value near 0 means no variation, a positive value means an improvement of the accuracy and a negative value means a worsening of the accuracy.

struggling to learn a combination of the two functions, this is also consistent with the fact that these two groups have a value of MSE_{old} very high, lower only than the one of $f_{prod,prod}$.
In the end, we can say that, if we are able to identify the hidden relation between two features, adding a new feature to the training set that represents this relationship can improve the performance of the ensemble. Besides, we highlight that this feature engineering process can be used iteratively, generating at the n-ith iteration a new feature that is possibly adequate complex to explain the entire ensemble.

Generating function	MSE_{old}	$R_c\%$	$MSE_{new,c}$	$\Delta_{MSE,c}\%$
$\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 + \mathcal{N}$	$1.75 \cdot 10^1$	100	$6.49 \cdot 10^0$	62.94
$(\mathbf{x}_1 + \mathbf{x}_2) * \mathbf{x}_3 + \mathcal{N}$	$1.26 \cdot 10^5$	59	$8.45 \cdot 10^4$	33.19
$(\mathbf{x}_1 + \mathbf{x}_2) / \mathbf{x}_3 + \mathcal{N}$	$2.29 \cdot 10^6$	1	$6.80 \cdot 10^2$	99.97
$\mathbf{x}_1 * \mathbf{x}_2 + \mathbf{x}_3 + \mathcal{N}$	$2.82 \cdot 10^4$	0	NA	NA
$\mathbf{x}_1 * \mathbf{x}_2 * \mathbf{x}_3 + \mathcal{N}$	$1.29 \cdot 10^9$	61	$5.92 \cdot 10^8$	54.14
$\mathbf{x}_1 * \mathbf{x}_2 / \mathbf{x}_3 + \mathcal{N}$	$9.54 \cdot 10^7$	14	$6.13 \cdot 10^7$	35.76
$\mathbf{x}_1 / \mathbf{x}_2 + \mathbf{x}_3 + \mathcal{N}$	$7.34 \cdot 10^4$	78	$9.31 \cdot 10^3$	87.31
$\mathbf{x}_1 / \mathbf{x}_2 * \mathbf{x}_3 + \mathcal{N}$	$3.00 \cdot 10^7$	24	$1.72 \cdot 10^7$	42.70
$\mathbf{x}_1 / \mathbf{x}_2 / \mathbf{x}_3 + \mathcal{N}$	$1.34 \cdot 10^4$	35	$7.45 \cdot 10^1$	99.44

Table 4.7: Average accuracy improvement when the relation between $\mathbf{x}_1$ and $\mathbf{x}_2$ was correctly identified. $R_c\%$ is the ratio of relationship correctly classified, $MSE_{new,c}$ represents the new MSE when the Miner Explainer Procedure was able to correctly identify the relationship between the features. The values in the column $\Delta_{MSE,c\%}$ have been computed as $\Delta_{MSE,c}\% = (MSE_{old} - MSE_{new,c})/MSE_{old} \cdot 100$ and they represent the accuracy variation in the new classifier with respect to the baseline, as in Table 4.6.

Generating function	MSE_{old}	$R_m\%$	$MSE_{new,m}$	$\Delta_{MSE,m}$ %
$\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 + \mathcal{N}$	$1.75e + 01$	0	NA	NA
$(\mathbf{x}_1 + \mathbf{x}_2) * \mathbf{x}_3 + \mathcal{N}$	$1.26 \cdot 10^5$	41	$1.32 \cdot 10^5$	-4.39
$(\mathbf{x}_1 + \mathbf{x}_2)/\mathbf{x}_3 + \mathcal{N}$	$2.29 \cdot 10^6$	99	$1.45 \cdot 10^6$	36.52
$\mathbf{x}_1 * \mathbf{x}_2 + \mathbf{x}_3 + \mathcal{N}$	$2.82 \cdot 10^4$	100	$1.00 \cdot 10^4$	64.41
$\mathbf{x}_1 * \mathbf{x}_2 * \mathbf{x}_3 + \mathcal{N}$	$1.29 \cdot 10^9$	39	$8.25 \cdot 10^8$	36.09
$\mathbf{x}_1 * \mathbf{x}_2/\mathbf{x}_3 + \mathcal{N}$	$9.54 \cdot 10^7$	86	$8.70 \cdot 10^7$	8.85
$\mathbf{x}_1/\mathbf{x}_2 + \mathbf{x}_3 + \mathcal{N}$	$7.34 \cdot 10^4$	22	$1.57 \cdot 10^5$	-114.70
$\mathbf{x}_1/\mathbf{x}_2 * \mathbf{x}_3 + \mathcal{N}$	$3.00 \cdot 10^7$	76	$3.24 \cdot 10^7$	-8.12
$\mathbf{x}_1/\mathbf{x}_2/\mathbf{x}_3 + \mathcal{N}$	$1.34 \cdot 10^4$	65	$8.79 \cdot 10^3$	34.40

Table 4.8: Average accuracy improvement when the relation between $\mathbf{x}_1$ and $\mathbf{x}_2$ was misclassified. $R_m\%$ is the ratio of relationship misclassified, $MSE_{new,m}$ represents the new MSE when the Miner Explainer Procedure was **not** able to correctly identify the relationship between the features. $\Delta_{MSE,m}\%$ have been computed in the same way as $\Delta_{MSE,c}\%$ in Table 4.7, but using $MSE_{new,m}$ and not $MSE_{new,c}$ as new accuracy value.

Generating function	MSE_{old}	$MSE_{new,o}$	$\Delta_{MSE,o}$
$\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 + \mathcal{N}$	$1.75 \cdot 10^1$	$6.49 \cdot 10^0$	62.94
$(\mathbf{x}_1 + \mathbf{x}_2) * \mathbf{x}_3 + \mathcal{N}$	$1.26 \cdot 10^5$	$1.04 \cdot 10^5$	17.78
$(\mathbf{x}_1 + \mathbf{x}_2)/\mathbf{x}_3 + \mathcal{N}$	$2.29 \cdot 10^6$	$1.44 \cdot 10^6$	37.15
$\mathbf{x}_1 * \mathbf{x}_2 + \mathbf{x}_3 + \mathcal{N}$	$2.82 \cdot 10^4$	$1.00 \cdot 10^4$	64.41
$\mathbf{x}_1 * \mathbf{x}_2 * \mathbf{x}_3 + \mathcal{N}$	$1.29 \cdot 10^9$	$6.83 \cdot 10^8$	47.10
$\mathbf{x}_1 * \mathbf{x}_2/\mathbf{x}_3 + \mathcal{N}$	$9.54 \cdot 10^7$	$8.34 \cdot 10^7$	12.62
$\mathbf{x}_1/\mathbf{x}_2 + \mathbf{x}_3 + \mathcal{N}$	$7.34 \cdot 10^4$	$4.19 \cdot 10^4$	42.87
$\mathbf{x}_1/\mathbf{x}_2 * \mathbf{x}_3 + \mathcal{N}$	$3.00 \cdot 10^7$	$2.88 \cdot 10^7$	4.07
$\mathbf{x}_1/\mathbf{x}_2/\mathbf{x}_3 + \mathcal{N}$	$1.34 \cdot 10^4$	$5.74 \cdot 10^3$	57.16

Table 4.9: Average accuracy improvement, using the Miner Explainer Procedure. The value in the column $\Delta_{MSE,o}\%$ have been computed as $\Delta_{MSE,o\%} = (MSE_{old} - MSE_{new,o})/MSE_{old} \cdot 100$ and they represent the accuracy variation in the new classifier with respect to the baseline, as in Table 4.6.

Results — Miner Explainer Procedure After having presented the results for the hypothetical scenario, where for all the features the right relationship classification was given, we describe the results when the Miner Explainer Procedure was used. To describe in detail the results obtained, we present them in three different tables: Table 4.7, Table 4.8, and Table 4.9. The three tables are used to represent respectively the difference of the accuracy when the procedure gave the correct classification, when the procedure gave a misclassification, and finally without distinguishing between correct and wrong classification. As we can see from Table 4.7, when the procedure has been able to correctly classify the interaction between $\mathbf{x}_1$ and $\mathbf{x}_2$, the resulting $\Delta_{MSE,c}$ has been always positive, indicating that the accuracy always improved at least of the 33.19% having as a baseline the accuracy without any feature engineering action employed MSE_{old}. On the other hand, considering the results when the predicted class was not correctly identified, we surprisingly see that for most of the generating function considered, the accuracy is improved in 5 cases over the 8 for which the $\Delta_{MSE,c}$ can be computed. This can be due to de fact that most of the errors involved the *div* and *prod*

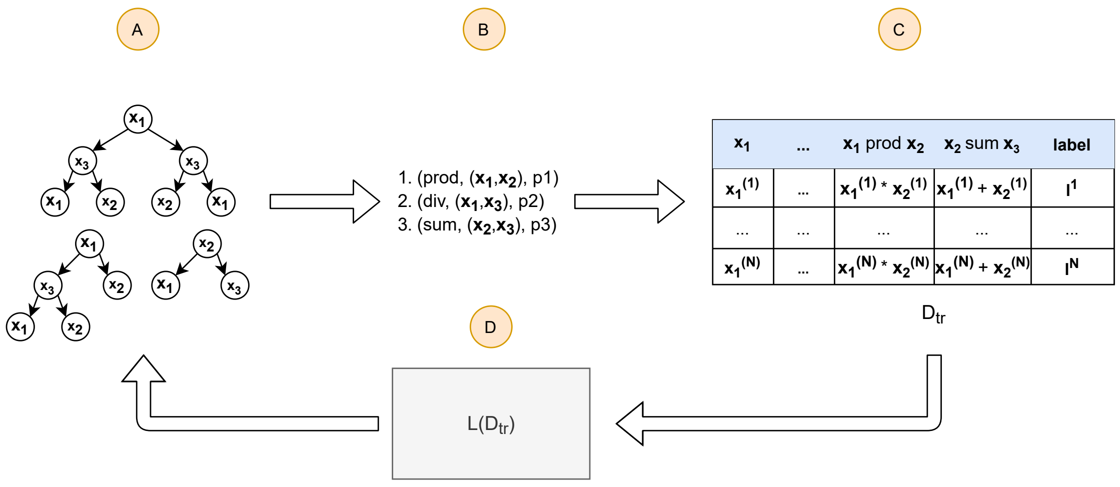


Figure 4.11: An illustrative example of the iterative procedure, summarized in four phases, from A to E, where the initial tree ensemble has been produced with the learning algorithm L. In A we have the ensemble to be explained, that in this example has only three features. Then, in phase B, we assume to have extracted all the possible relations between the features via Algorithm 4, that are represented as triples, where the first element is the type of relation found, the second is a couple containing the feature involved, and the third is the probability of that relation given by the classifier. In C, we visually present the operation of augmenting the training dataset D_{tr} with the new relations found, sorted by their probability, and considering only the ones that improve the accuracy of the classifier. Finally, in D we create a new ensemble with the learning algorithm L, and then we repeat the whole procedure from A.

classes, and adding a new feature that represent a division instead of a multiplication can improve the performance in spite of the classification error. Finally, from Table 4.9 we can say that the accuracy is improved on average on every dataset under investigation.

Iterating the procedure To conclude the analysis on the performance improvement, we have reiterated the process to find other relations after adding the fourth feature $\mathbf{x}_4$ to the dataset. We recall that the feature $\mathbf{x}_4$ has been added to the dataset to represent the relationship between $\mathbf{x}_1$ and $\mathbf{x}_2$, thus, for example, for the generating function $f_{sum,prod}(\mathbf{x})$, if we correctly classify the relationship between the two features under investigation we would have added $\mathbf{x}_4 = \mathbf{x}_1 + \mathbf{x}_2$ to our dataset. In particular, after the addition of $\mathbf{x}_4$, we have investigated the relationship between the features $\mathbf{x}_4$ and $\mathbf{x}_3$ to confirm our hypothesis about the composition of the patterns. Following our hypothesis, if we consider for example $f_{sum,prod}(\mathbf{x}) \simeq (\mathbf{x}_1 + \mathbf{x}_2) \cdot \mathbf{x}_3$, after adding $\mathbf{x}_4 = \mathbf{x}_1 + \mathbf{x}_2$ we would have expected to find a *prod* relationship between $\mathbf{x}_4$ and $\mathbf{x}_3$, and this would have lead us to create a fifth feature $\mathbf{x}_5 = \mathbf{x}_4 \cdot \mathbf{x}_3$, that would have represented the entire function learned by the ensemble.
However, this was not the case, and after the addition of $\mathbf{x}_4$ in all the datasets from each generating function, the relation between $\mathbf{x}_4$ and $\mathbf{x}_3$ has always been classified as *random* by the classifier created. This unexpected result is due to the fact there were very few frequent subtrees containing both $\mathbf{x}_4$ and $\mathbf{x}_3$, and even with a smaller *minsup* a lot of frequent subtrees only made of $\mathbf{x}_3$ and few with $\mathbf{x}_4$ were present.

However, the results presented in chapter 5 with real-world dataset showed that reiterating the procedure can improve the performance of the classifier, although the poor results obtained with the synthetic datasets. In Figure 4.11 there is a visual representation of the **general** iterative procedure proposed, where at each iteration a new ensemble is created over an augmented dataset with features that represent the relations discovered through the frequent pattern mining. The entire procedure is explained in detail in chapter 5.

4.7 Conclusion and summary

Conclusion In this chapter, we have investigated a possible path to explain a tree ensemble via pattern discovery, and in particular via frequent subtree mining. We have formulated our investigation as 4 main research questions, that describe our main goal, i.e. find a closed formed expression that accurately approximates the function learned by the ensemble, and other sub-goals that can be also interesting per se. We have tried to search an answer for RQ 2, RQ 3, RQ 4 using synthetic datasets. With our experiments we have shown that there are strong indications that we have an affirmative answer for RQ 3 and RQ 4, meaning that frequent patterns found in the ensemble are strongly related to the function learned, and using this information can greatly improve the performance of the ensemble. However, there are is no strong evidence suggesting that the frequent subtrees found for simple functions can be used to find the interaction between features in more complex functions. Finding the presence or absence of this relation could be the last step to approach a solution to our initial research question (RQ 1) and thus find an approximation of the function learned by the ensemble via a closed-form expression.

Summary The summary of the chapter divided by sections is the following:

- In section 4.1 we have presented the main research questions of this thesis, that describe what we wanted to investigate in this thesis. Summing up, we would like to find a possible answer to the model explanation problem for tree ensembles through pattern discovery.

- In section 4.2 there are described the main characteristics of the frequent subtree mining problem, and we have presented the two main algorithms that have been used in the development of this thesis, namely SLEUTH and CMTreeMiner.

- In section 4.3 we have presented the main framework that we have developed to analyze the interaction between features and, if used iteratively, try to solve the model explanation problem.

- In section 4.4 we have presented an analysis over synthetic datasets formed by two features to discover if specific frequent subtrees are in some way correlated with the type of feature's relationship.

- In section 4.5 we have analyzed in synthetic datasets composed of three features if it is possible to find a specific type of relationship exploiting the information from the analysis of the frequent subtrees between two features.

- In section 4.6 we have analyzed the performance improvement for an ensemble trained over an augmented dataset, where a new feature is added to represent a relation between features through frequent subtree mining.

Chapter 5

Evaluation with real-world datasets

In this chapter, we present an evaluation of the method proposed in chapter 4 with three real-world datasets provided by the University of California Irvine (UCI) machine learning repository [16]. Since there are no standard or well-known benchmarks to evaluate the efficacy of an explanation method of this type, we also propose our quantitative evaluation methodology based on the improvement of the accuracy of an ensemble. Finally we discuss the results found and what is missing from this approach that can be useful to be implemented in the future.

5.1 Datasets

To perform this evaluation over real-world dataset we have chosen three datasets from the UCI data repository following some simple criteria:

1 The default task for the dataset must be a regression problem.

2 The majority of the features in the dataset must be numerical.

3 The dataset must have a small number of features, i.e. if M is the number of features of D, $M \leq 20$.

The first criterion is based on the fact that in this work we have focused only on regression forest, and so we leave the investigation over classification forest as future work. The second criterion is also based on what we have analyzed in this thesis, and thus we preferred to focus only on numerical features, even though the whole investigation can be extended to categorical features. The last criterion has been chosen to limit the number of features used by the tree ensemble since we have seen from the analysis in the synthetic datasets that the greater the number of features involved, the lower the number of frequent subtrees when the ensemble is formed by a fixed number of trees. This behavior suggest to use an adaptive *minsup* to modulate, according to the forest under consideration, the subtrees considered. To deal with this problem also in datasets with a small number of features, in our evaluation we chose to select three different *minsup*, and analyze the results obtained. Eventually, the following datasets have been chosen: the Bike Sharing Dataset, the Airfoil Self-Noise Data Set, and the Wine Quality Data Set. The full description of each dataset is presented in the following paragraphs.

Bike Sharing Dataset The Bike Sharing Dataset has been already presented in section 3.1, and in this part of the analysis, only a small part of its features have been used to avoid dealing with categorical features, or features containing dates. In this way we have transformed the problem on finding the number of bikes rented per day using only three features, namely the temperature (*tmp*), the humidity (*hum*), and the wind speed (*windspeed*). Since the feature space in this dataset has been over-simplified, it has to be seen only as a working example, with real-world data and within a very low dimensional space.

Airfoil Self-Noise Data Set The Airfoil Self-Noise Data Set is a dataset donated in the 2014 to the UCI Machine Learning Repository that was used from the National Aeronautics and Space Administration (NASA) to predict the noise of a specific airfoil (NACA 0012) at different wind tunnel speeds and angles of attack. The features used are described in Table 5.1.

Feature Name	Description
freq	Frequency of the sound wave, in Hertz.
angle	Angle the wind attack to the airfoil, in degree.
chord	Chord length: specific measure of the airfoil, in meter.
vel	Velocity of the wind speed under consideration, in meter per second.
ssdt	Suction sid displacement thickness, in meters.
spl	Scaled Sound Pressure Level (SPL), in decibels. Used as target for the regression problem.

Table 5.1: Features of the Airfoil Self-Noise Data Set.

Wine Quality dataset The last dataset used is the well-known wine quality dataset[14], which is used to predict the quality of the Portuguese "Vinho Verde" wine — with a score between 0 and 10 — given some physicochemical properties. The dataset is divided into two sub-datasets, one containing only red wines and the other wine containing only white wines. In each sub-datasets, there are 11 features that can be used to predict the score associated with each wine and are listed in Table 5.2.

5.2 Evaluation methodology

To evaluate the application of the techniques presented in the previous chapter in real-world scenarios, we propose to check the improvement of specific evaluation metrics after having augmented the dataset with new features discovered through frequent subtree mining. In particular, we propose an iterative approach, where at each iteration we apply Algorithm 4 for each possible pair of features, and if a relation is found, we augment the dataset and we learn again a new tree ensemble if it improves its MSE.
In this section, we first review the metrics involved, and then we describe in detail the procedure employed.

Feature Name	Description
f_acidity	Fixed acidity, measured in gtartaric acid)/dm^3
v_acidity	Volatile acidity, measured in g(acetic acid)/dm^3
c_acid	Citric acid, measured in g/dm^3
r_sugar	Residual sugar, measured in g/dm^3
chlorides	Chlorides, measured in g(sodium chloride)/dm^3
f_s_dioxide	Free sulfur dioxide, measured in mg/dm^3
t_s_dioxide	Total sulfur dioxide, measured in mg/dm^3
density	g/cm^3
ph	Measured pH of the wine.
sulphates	Sulphates found in the wine, measured in g(potassium sulphate)/dm3^3.
alcohol	Alcohol level of the wine, measured in vol.%.
quality	Quality of the wine, target of the regression problem. It is a score from 0 to 10.

Table 5.2: Features of the Wine Quality dataset.

The metrics The two metrics used to evaluate the techniques under investigation are the MSE, and the R^2. With the MSE, that has been defined previously, we compute the average squared error loss on a given test set. Instead, with R^2 (R squared) we indicate the coefficient of determination, and it represents the amount of variance of y explained by the ML model. Let D be the test dataset where we want to compute the R^2 being composed of N samples, and $\bar{y}$ the average label value in the dataset, i.e. $\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$, the estimation of the R^2 score is defined as follows:

$$R^2(D) = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y_i})^2}{\sum_{i=1}^{N}(y_i - \bar{y_i})^2}$$

From the equation above, it easy to see that the maximum score that we can obtain is 1, in fact, when we get a $R^2(D) = 1$ it means that no error has been made over the test set.

The procedure First of all, we divide the dataset into three parts, training set D_{tr}, validation set D_v, and test set D_{te}. In our case, we have divided the dataset into two parts: 70% for the training phase and the validation phase, and 30 % for the test phase. The former has been further divided in 80-20% to create the training and the validation sets. Summing up, the 56% has been used as training set, the 14 % as validation set, and the 30 % as test set. Then, we compute our baseline, that is the MSE and the R^2 obtained by an ensemble E, learned with the learning algorithm L over D_{tr}, trying to predict the labels in D_{te}.
After having computed the baseline, we apply the procedure Algorithm 4 to all the possible pairs (l_1, l_2), where $l_1, l_2 \in F$ and F is the set of all the features under investigation. Then we create a list R, containing a triple composed by three elements: the relation type rel found thanks to Algorithm 4, the pair of features involved (l_1, l_2), and the probability $P(rel)$ that the Algorithm 4 procedure has associated with rel.
Given the list R, we sort it by the probability of each relation rel found. Subsequently, we iterate over each element of R, and we augment the datasets with the new relation found adding the corresponding feature, similar to what we have

done in section 4.6, creating a new dataset D'_{tr}. The procedure in Algorithm 5, is called $augment(D_{tr}, rel, (l_1, l_2))$.
At each iteration, we learn a new ensemble E' over D'_{tr} with the learning algorithm L, and if the MSE improves over the validation set, we substitute E with E', and the training and validation set accordingly. We repeat this procedure until there is an improvement of the performance on the validation set. The full procedure is presented as pseudocode in Algorithm 5 using also a sub-procedure, presented in Algorithm 6 for readability purposes.

Algorithm 5 Test performance improvement via MinerExplainer

```
1: procedure MINEREXPLAINERTEST($D_{tr}, D_v, L, M, C$))
2:     ▷ Training dataset $D_{tr}$, Validation dataset $D_v$, Tree ensemble learning algorithm $L$, subtree miner algorithm $M$, and the relationship classifier $C$
3:     $E \leftarrow L(D_{tr})$
4:     do
5:         $p_{old} \leftarrow evaluate(E, D_v)$                ▷ Get performance measure
6:         $p \leftarrow p_{old}$
7:         $R \leftarrow MinerExplainerAll(D_{tr}, E, M, C)$
8:         $sort(R)$                                             ▷ Sort $R$ by $P(rel)$
9:         for each $(rel, (l_1, l_2), P(rel)) \in R$ do
10:            $D'_{tr}, D'_v \leftarrow augment(D_{tr}, rel, (l_1, l_2)), augment(D_v, rel, (l_1, l_2))$
11:            $E' \leftarrow L(D'_{tr})$
12:            $p' \leftarrow evaluate(E', D'_v)$
13:            if $p'$ is better than $p$ then
14:                $D_{tr}, D_v, E \leftarrow D'_{tr}, D'_v, E'$
15:                $p \leftarrow p'$
16:            end if
17:        end for
18:    while $p$ is better than $p_{old}$
19:    return $E$
20: end procedure
```

Finally, after having applied the Algorithm 5, we compare the resulting MSE and R^2 of the new ensemble created with respect to the baseline.

It is worth noticing that the procedure described above has a underling assumption: when the accuracy of the regression model improves, it means that the feature added really represents an hidden relationships between features that the ensemble has tried to model. If this be eventually proved correct, we can say that with this procedure we both explain and improve the model.

5.3 Results

We applied the evaluation methodology proposed in the previous section to the three datasets (four considering the subdivision of the wine dataset) for ensemble learned via GBDT or via RF. For each ensemble learning algorithm, we have used the default parameters proposed by the LGBM package, except for: the learning rate in LGBM that has been set through a ten-fold cross validation over the training and validation set between the possible choices $\{0.1, 0.01, 0.001\}$; and the

Algorithm 6 Sub procedure for MinerExplainerTest

```
1: procedure MinerExplainerAll($D_{tr}, E, M, C$)
2:     ▷ Training dataset $D_{tr}$, tree ensemble $E$, subtree miner algorithm $M$, and the relationship classifier $C$
3:     $\mathcal{S} \leftarrow Comb(D_{tr}, 2)$        ▷ Get all the possible couple of features
4:     $L \leftarrow list()$        ▷ Empty list
5:     for each $(l_1, l_2) \in \mathcal{S}$ do
6:         $rel \leftarrow MinerExplainer(E, M, C, (l_1, l_2))$
7:         if $rel \neq rnd$ then
8:             $t \leftarrow (rel, (l_1, l_2), P(rel))$
9:             $L.append(t)$
10:        end if
11:    end for
12:    return $L$
13: end procedure
```

three parameters for RF that must be set manually, namely the bagging frequency that has been set to 2, the bagging fraction that has been set to 100% and the feature fraction that has been set to 70%. In addition, for each dataset we tried three different *minsup* for the miner M used in Algorithm 5: 75%, 50%, and 30%. We have applied this strategy because the number of frequent subtrees varies a lot from dataset to dataset, and, for example, in the wine quality dataset using GBDT, there were practically no frequent subtrees with $minsup = 75$ and $minsup = 50$. However, this can be seen as a not-so-sensible strategy, because the classifier C used in Algorithm 5 has been trained over frequent subtree mined only $minsup = 75$. Even though we acknowledge this possible inconsistency in the procedure, we also highlight that creating a dataset with subtrees having $minsup < 75$ with an adequate number of samples to train a new classifier C, was practically infeasible with the computational resource available. Notwithstanding this possible bias in the feature relation classification, we tried this possible setting to see if the procedure can still give an improvement in performance.

In this section we first discuss the results obtained with GBDT, then the ones obtained with RF, and finally we present in depth the results of a specific dataset with a fixed *minsup* as a case study.

Gradient Boosting Decison Tree The results obtained from the evaluation over the three datasets for ensembles learned via GBDT are summarized in Table 5.3. As we can see from the table with the test procedure proposed we always get an improvement in the performance of each regressor, except for the one trained over the red wine quality dataset. This can be seen as a good achievement also without the interpretation of the new feature added to each dataset, and we highlight that we got a 34% improvement in accuracy with respect to the initial MSE in the airfoil self-noise dataset, proving that a feature engineering process, as the one proposed, can greatly improve the performance of a tree ensemble. From the results obtained we can also see that the lower the *minsup* used, the greater the number of relations are discovered. Lowering the *minsup* can be a possible solution to find more frequent subtrees but can possibly result in finding spurious

Dataset	MSE	R^2	$minsup\%$	K	MSE_{miner}	R^2_{miner}
Bike sharing	1.88×10^6	0.53	75	1	1.85×10^6	0.53
			50	1	1.82×10^6	0.54
			30	1	1.85×10^6	0.53
Airfoil Self-Noise	4.29×10^0	0.91	75	3	3.53×10^0	0.93
			50	3	3.34×10^0	0.93
			30	21	2.81×10^0	0.94
Wine quality - red	3.5×10^{-1}	0.44	75	0	3.5×10^{-1}	0.44
			50	0	3.5×10^{-1}	0.44
			30	7	3.6×10^{-1}	0.43
Wine quality - white	4.3×10^{-1}	0.43	75	0	0.43×10^0	0.43
			50	0	4.3×10^{-1}	043
			30	2	4.1×10^{-1}	0.45

Table 5.3: Table containing the results of the evaluation proposed using GBDT as learning algorithm. The columns MSE and R^2 are referred to the results obtained with the initial, unmodified tree ensemble, and MSE_{miner} and R^2_{miner} represents the results obtained with the modified ensemble according to Algorithm 5. Finally, the *minsup* column contains the values of *minsup* (in percentage) used to mine the subtrees from the various ensembles, and K represents the number of features added to each datasets. Best results for each dataset are highlighted in green.

relations that give no improvement to the forest performance, as for the red wine quality dataset with $minsup = 30$.

Random Forest To perform the analysis also with ensembles learned by RF, we create a new relation classifier C, employing the same procedure described in chapter 4, but using the RF learning algorithm instead of GBDT. The results obtained are summarized in Table 5.4. From the outcomes showed in the table, it is worth to highlight we have an improvement of the performance in each dataset, but the number of features added is significantly higher with respect to the gradient boosting case. This can be explained by the fact that in RF the majority of the trees do not differ to much as for GBDT, because each tree in the ensemble do not account for the errors made by other trees, and thus they are not so dependent from each other.
We also notice that the highest improvement in performance is obtained, as for the GBDT analysis, over the airfoil self-noise dataset, with an improvement of almost 61% with respect to the baseline.

Case study To give an example of the feature added to the dataset, we analyze the results during the procedure for the white wine quality dataset where *minsup* was set to 30 and the learning algorithm GBDT were used. At the first iteration, three tuples were created with the sub-procedure presented in Algorithm 6 (we simplify the identification of the features using their name as presented in Table 5.2):

$$\begin{aligned} t_1 =& (div, (f\_s\_dioxide, t\_s\_dioxide), 0.99) \\ t_2 =& (sum, (f\_s\_dioxide, alcohol), 0.78) \\ t_3 =& (sum, (t\_s\_dioxide, alcohol), 0.78) \end{aligned}$$

Dataset	MSE	R^2	$minsup\%$	K	MSE_{miner}	R^2_{miner}
Bike sharing	1.89×10^6	0.52	75	4	1.85×10^6	0.53
			50	7	1.86×10^6	0.53
			30	0	1.89×10^6	0.52
Airfoil Self-Noise	1.74×10^1	0.63	75	12	8.86×10^0	0.82
			50	27	8.48×10^0	0.82
			30	55	7.04×10^0	0.85
Wine quality - red	4.04×10^{-1}	0.36	75	8	4.01×10^{-1}	0.37
			50	9	3.96×10^{-1}	0.37
			30	17	4.00×10^{-1}	0.37
Wine quality - white	4.86×10^{-1}	0.36	75	7	4.78×10^{-1}	0.36
			50	17	4.81×10^{-1}	0.36
			30	13	4.79×10^{-1}	0.37

Table 5.4: Table containing the results of the evaluation proposed using RF as learning algorithm. The columns and rows have the same meaning as in Table 5.3.

Then, they have been added in order, i.e. first t_1, then t_2, and finally t_3, to the dataset, and each time the MSE has been computed with respect to the validation dataset D_v. Only t_1 and t_3 result in increasing the accuracy over D_v and, thus they are the two features added to at the end of the first iteration. At the second iteration, only one tuple was created by Algorithm 6, that is exactly t_2, that was not added to the dataset because no improvement was found over the validation set also in the second iteration. Therefore, at the end of the procedure only two features where added: $\mathbf{x}_{11} = \frac{f\_s\_dioxide}{t\_s\_dioxide}$ and $\mathbf{x}_{12} = t\_s\_dioxide + alcohol$. This can be seen as a first step versus a possible answer to the RQ 1, in fact we can say that if the classifier C has correctly identify the relation between the feature involved the closed-form expression is function also of the new two feature added: $f_e(\mathbf{x}) = f_e(\mathbf{x}_{11}, \mathbf{x}_{12}, ...)$. We also highlight that the three relations discovered can also possibly have a meaningful interpretation. For example, t_1 represents a ratio between the mg/dm^3 of free sulfur dioxide and the total sulfur dioxide, that are indeed related. In addition, at the second iteration, when the features $\mathbf{x}_{11}$ and $\mathbf{x}_{12}$ have been already added, we do not find again the relation represented by t_3 and t_1 because they were already added to the dataset. This means that when a relation between tow features is added to the dataset, the learning algorithm can exploit it, and so the number of frequent pattern that involve the two specific features decrease.

5.4 Conclusion and summary

Conclusion In this chapter, we proposed an evaluation strategy for the forest explanation techniques through a quantitative evaluation of the accuracy improvement. We evaluate multiple tree ensemble models learned via GBDT and RF over three real-world datasets. The result obtained suggest that, in the majority of the cases, we can significantly improve the accuracy of a tree ensemble using the information retrieved from its frequent tree-based pattern. In addition, through a case study, we showed also that the relations found could have a meaningful interpretation, and can be used as a first step to solve the model explanation problem.

Finally, it is worth noticing that we have not compared the results obtained with any other methods of the current state of the art because there is no such technique that does a similar explanation as we propose. As we have presented in chapter 3, other popular methods, such as SHAP or LIME, do not take into account the type of relationship between the features, and they are not focused on improving the performance of the model; they are instead focused on justify the outcome of a model. We consider that shifting the goal of the explanation method from *explain to justify* to *explain to improve* (see section 2.3) to be a step towards moving the evaluation to a more quantitative approach, rather than qualitative, that can be valid for all the methods. Having a metric that can be compared among other explanation techniques, such as the accuracy of the model, can be seen as a common benchmark to further investigations of this type.

Summary The summary of the chapter divided by sections is the following:

- In section 5.1 we have presented the three datasets under consideration, namely the bike sharing dataset, the airfoil self-noise dataset, and the wine quality dataset.
- In section 5.2 we have illustrated the evaluation strategy proposed to check the improvement of accuracy in a tree ensemble through pattern discovery. The underling assumption is that if the accuracy improved, the relation found via the tree-base patterns are also the relation learned by the ensemble.
- In section 5.3 we have discussed the results obtained over the three datasets and with the two learning methods of choice. In addition, a case study is presented to better illustrate the method proposed.

Chapter 6

Conclusions

In this conclusive chapter we summarize the results obtained, we highlight the main limitation and drawbacks of the proposed approach, and we describe the possible future works that can be done to improve the framework and to tackle its limitations.

6.1 Conclusions

In this thesis, we have begun the evaluation of a new technique to explain a tree ensemble through frequent pattern discovery. The results obtained both on synthetic datasets and in real-world datasets suggest that there is a correlation between frequent subtrees and the function learned by the model that can be exploited. Specifically, we have proposed a framework to express the problem of finding the pairwise relation learned by a tree ensemble between features as a classification problem. In this setting, every relation between features is represented by the frequent subtrees retrieved from the forest under investigation. To get a good classification accuracy, the subtrees mined have to be selected, filtered, and manipulated to properly represent a specific relation. Besides, we propose to iterate this process adding at each iteration a new feature representing the interaction discovered, to improve the accuracy of the ensemble and create more complex features that can eventually represent the entire function learned. To evaluate the feasibility of this approach we have divided our investigation into three parts: we have analyzed the frequent subtree in tree ensemble learned by dataset generated by simple algebraic functions composed only by two features, we have investigated the possibility to extend the approach to function composed by three features, and we have measured the impact of this approach on the accuracy of the model in synthetic datasets.
After the evaluation over synthetic datasets, it is still not clear if it is correct to assume that the frequent patterns found in an ensemble that mimics simple functions can be used to classify the interaction between features in more complex ensembles. As suggested from the results obtained over the synthetic datasets, that are some indications of a possible affirmative answer to our hypothesis but they are not completely satisfactory. What is clear, is that the feature engineering process used can improve the accuracy of the model also in real-world datasets as we have shown in chapter 5.
However, there is still work to do to understand if the relations found are not spurious and how the improvement of the accuracy is related to the feature engineering

process.
In addition, there are also limitations in the performance with the mining algorithm used, which can be probably overcome with further and better implementations. For example, it would be helpful to implement a parallel version of a frequent subtree mining algorithm for both ordered and unordered trees, since the candidate generation in the mining algorithms used is suitable to be parallelized.

6.2 Future works

Since what has been presented is a totally novel approach, there are plenty of possible ways that can be undertaken to improve it and extend its applicability.
Specifically, in this work, we have only considered forests that solve regression problems with continuous features, but it would be interesting to extend the approach proposed also to tree ensemble for classification and with categorical features.
Furthermore, it could be sensible to try different approaches to classify the relation between features from the one proposed in this thesis. For example, to speed up the process and to avoid finding spurious relations, we might want to identify the relationship starting from the known patterns and count the matches in the forest, i.e. find isomorphism of patterns, that we know are associated with a particular relation and classify the features involved accordingly.
Finally, it would also be possible to extend the analysis of frequent patterns to other ML models based on graphs, such as Deep Neural Network, mining frequent subgraphs to identify specific feature interactions.

Acknowledgement

This thesis would not have been accomplished without the support of a lot of people, both inside and outside the University, during the entire period of my studies.

First of all, I would like to thank my supervisor, Professor Claudio Lucchese for having introduced me to the interesting research topic discussed in this thesis, and for his invaluable help to formulate the research questions and the methodology proposed to solve them.
I would like also to thank Professor Alessandra Raffaetà, that gave me precious help during my study abroad at the Eindhoven University of Technology, which has been a fundamental part of my experience during my master's degree.

Naturally, I thank my whole family for the incredible support and to have always believed in me. Nothing of what I have done during these years would have been possible without them.

I would also thank all my friends, that are essential in my life and from whom I have been apart for a long time during this crazy year. I hope that the ongoing pandemic will end soon and we will return to hang out together as usual.

Last but not least, I send a lovely thank you to Elisa, that has been extremely patient with me during these years especially during my absence for my studies abroad, and she has been able to see the man that I am, and the man that I would like to be.

This thesis is entirely dedicated to my grandmother that a few days ago has left us. She will always be in our minds and hearts, especially on this occasion, given her pride in her grandsons' academic achievements. Ti mando un ultimo abbraccio nonna.

Bibliography

[1] Amina Adadi and Mohammed Berrada. Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI). *IEEE Access*, 6:52138–52160, 2018. ISSN 2169-3536. doi: 10.1109/ACCESS.2018.2870052.

[2] Saleema Amershi, Maya Cakmak, William Bradley Knox, and Todd Kulesza. Power to the People: The Role of Humans in Interactive Machine Learning. *AI Magazine*, 35(4):105–120, December 2014. ISSN 2371-9621. doi: 10.1609/aimag.v35i4.2513. Number: 4.

[3] Yali Amit and Donald Geman. Randomized Inquiries About Shape: An Application to Handwritten Digit Recognition. Technical report, CHICAGO UNIV IL DEPT OF STATISTICS, November 1994. Section: Technical Reports.

[4] Daniel W. Apley and Jingyu Zhu. Visualizing the effects of predictor variables in black box supervised learning models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 82(4):1059–1086, September 2020. ISSN 1369-7412, 1467-9868. doi: 10.1111/rssb.12377.

[5] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58:82–115, June 2020. ISSN 15662535. doi: 10.1016/j.inffus.2019.12.012.

[6] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, August 1996. ISSN 0885-6125, 1573-0565. doi: 10.1007/BF00058655.

[7] Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, October 2001. ISSN 1573-0565. doi: 10.1023/A:1010933404324.

[8] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[9] Nick Cammarata, Shan Carter, Gabriel Goh, Chris Olah, Michael Petrov, and Ludwig Schubert. Thread: Circuits. *Distill*, 5(3):e24, March 2020. ISSN 2476-0757. doi: 10.23915/distill.00024.

[10] Hong Cheng, Xifeng Yan, Jiawei Han, and Chih-Wei Hsu. Discriminative Frequent Pattern Analysis for Effective Classification. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 716–725, Istanbul, April 2007. IEEE. ISBN 978-1-4244-0802-3. doi: 10.1109/ICDE.2007.367917.

[11] Yun Chi, Richard R. Muntz, Siegfried Nijssen, and Joost N. Kok. *Frequent Subtree Mining - An Overview*. Ios press, 2005.

[12] H A Chipman, E I George, and R E McCulloch. Making sense of a forest of trees. *Computing Science and Statistics*, page 10, 1998.

[13] Paulo Cortez and Mark J. Embrechts. Opening black box Data Mining models using Sensitivity Analysis. In *2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, pages 341–348, Paris, France, April 2011. IEEE. ISBN 978-1-4244-9926-7. doi: 10.1109/CIDM.2011.5949423.

[14] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553, November 2009. ISSN 01679236. doi: 10.1016/j.dss.2009.05.016. URL `https://linkinghub.elsevier.com/retrieve/pii/S0167923609001377`.

[15] Pedro Domingos. Knowledge Discovery Via Multiple Models. *Intelligent Data Analysis*, 2 (1-4):16, 1998.

[16] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

[17] Hadi Fanaee-T and Joao Gama. Event labeling combining ensemble detectors and background knowledge. *Progress in Artificial Intelligence*, 2(2):113–127, June 2014. ISSN 2192-6360. doi: 10.1007/s13748-013-0040-3.

[18] Yoav Freund and Robert E Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1): 119–139, August 1997. ISSN 00220000. doi: 10.1006/jcss.1997.1504.

[19] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232, October 2001. ISSN 0090-5364, 2168-8966. doi: 10.1214/aos/1013203451. Publisher: Institute of Mathematical Statistics.

[20] Jerome H. Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, February 2002. ISSN 01679473. doi: 10.1016/S0167-9473(01)00065-2.

[21] Alex Goldstein, Adam Kapelner, Justin Bleich, and Emil Pitkin. Peeking Inside the Black Box: Visualizing Statistical Learning with Plots of Individual Conditional Expectation. *arXiv:1309.6392 [stat]*, March 2014. arXiv: 1309.6392.

[22] Bryce Goodman and Seth Flaxman. European union regulations on algorithmic decision-making and a "right to explanation". *AI Magazine*, 38(3):50–57, October 2017. ISSN 2371-9621, 0738-4602. doi: 10.1609/aimag.v38i3.2741.

[23] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A Survey of Methods for Explaining Black Box Models. *ACM Computing Surveys*, 51(5):1–42, August 2018. ISSN 03600300. doi: 10.1145/3236009.

[24] David Gunning and David Aha. DARPA's Explainable Artificial Intelligence (XAI) Program. *AI Magazine*, 40(2):44–58, June 2019. doi: 10.1609/aimag.v40i2.2850.

[25] David Gunning, Mark Stefik, Jaesik Choi, Timothy Miller, Simone Stumpf, and Guang-Zhong Yang. XAI-Explainable artificial intelligence. *Science Robotics*, 4(37):eaay7120, December 2019. ISSN 2470-9476. doi: 10.1126/scirobotics.aay7120.

[26] Robert R. Hoffman and Gary Klein. Explaining Explanation, Part 1: Theoretical Foundations. *IEEE Intelligent Systems*, 32(3):68–73, May 2017. ISSN 1541-1672. doi: 10.1109/MIS.2017.54.

[27] Robert R. Hoffman, Shane T. Mueller, and Gary Klein. Explaining Explanation, Part 2: Empirical Foundations. *IEEE Intelligent Systems*, 32(4):78–86, 2017. ISSN 1541-1672. doi: 10.1109/MIS.2017.3121544.

[28] Aída Jiménez, Fernando Berzal, and Juan-Carlos Cubero. Frequent tree pattern mining: A survey. *Intelligent Data Analysis*, 14(6):603–622, November 2010. ISSN 15714128, 1088467X. doi: 10.3233/IDA-2010-0443.

[29] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. *Advances in Neural Information Processing Systems*, 30, 2017.

[30] R. H. Kewley, M. J. Embrechts, and C. Breneman. Data strip mining for the virtual design of pharmaceuticals with neural networks. *Trans. Neur. Netw.*, 11(3):668–679, May 2000. ISSN 1045-9227. doi: 10.1109/72.846738.

[31] Gary Klein. Explaining Explanation, Part 3: The Causal Landscape. *IEEE Intelligent Systems*, 33(2):83–88, March 2018. ISSN 1541-1672. doi: 10.1109/MIS.2018.022441353.

[32] Sanjay Krishnan and Eugene Wu. PALM: Machine Learning Explanations For Iterative Debugging. In *Proceedings of the 2nd Workshop on Human-In-the-Loop Data Analytics*, pages 1–6, Chicago IL USA, May 2017. ACM. ISBN 978-1-4503-5029-7. doi: 10.1145/3077257.3077271.

[33] Wei-Yin Loh. Classification and regression trees. *WIREs Data Mining and Knowledge Discovery*, 1(1):14–23, January 2011. ISSN 1942-4787, 1942-4795. doi: 10.1002/widm.8.

[34] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 4768–4777, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

[35] Scott M. Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M. Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence*, 2(1):56–67, January 2020. ISSN 2522-5839. doi: 10.1038/s42256-019-0138-9. Number: 1 Publisher: Nature Publishing Group.

[36] David J C MacKay. *Information Theory, Inference, and Learning Algorithms.* Cambridge University Press, 2005.

[37] Sina Mohseni, Niloofar Zarei, and Eric D. Ragan. A Multidisciplinary Survey and Framework for Design and Evaluation of Explainable AI Systems. *arXiv:1811.11839 [cs]*, August 2020. arXiv: 1811.11839.

[38] Christoph Molnar. *Interpretable Machine Learning.* 2019. `https://christophm.github.io/interpretable-ml-book/`.

[39] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[40] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, March 1986. ISSN 0885-6125, 1573-0565. doi: 10.1007/BF00116251.

[41] J Ross Quinlan. *C4. 5: programs for machine learning.* Elsevier, 2014.

[42] John R Quinlan et al. Learning with continuous classes. In *5th Australian joint conference on artificial intelligence*, volume 92, pages 343–348. World Scientific, 1992.

[43] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144, San Francisco California USA, August 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939778.

[44] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, May 2019. ISSN 2522-5839. doi: 10.1038/s42256-019-0048-x.

[45] Stuart Russell and Peter Norvig. *Artificial intelligence: a modern approach - 4th edition.* Pearson, 2020.

[46] Omer Sagi and Lior Rokach. Ensemble learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8(4), July 2018. ISSN 1942-4787, 1942-4795. doi: 10.1002/widm.1249.

[47] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, June 1990. ISSN 1573-0565. doi: 10.1007/BF00116037.

[48] Lloyd S. Shapley and Alvin E. Roth, editors. *The Shapley value: essays in honor of Lloyd S. Shapley.* Cambridge University Press, Cambridge [Cambridgeshire] ; New York, 1988. ISBN 978-0-521-36177-4.

[49] Keng Siau and Weiyu Wang. Building Trust in Artificial Intelligence, Machine Learning, and Robotics. *Cutter Business Technology Journal*, page 8, 2018.

[50] Dylan Slack, Sophie Hilgard, Emily Jia, Sameer Singh, and Himabindu Lakkaraju. Fooling LIME and SHAP: Adversarial Attacks on Post hoc Explanation Methods. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, pages 180–186, New York NY USA, February 2020. ACM. ISBN 978-1-4503-7110-0. doi: 10.1145/3375627.3375830.

[51] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, August 1998. ISSN 01628828. doi: 10.1109/34.709601.

[52] Anneleen Van Assche and Hendrik Blockeel. Seeing the Forest Through the Trees: Learning a Comprehensible Model from an Ensemble. In Joost N. Kok, Jacek Koronacki, Raomon Lopez de Mantaras, Stan Matwin, Dunja Mladenič, and Andrzej Skowron, editors, *Machine Learning: ECML 2007*, volume 4701, pages 418–429. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-74957-8 978-3-540-74958-5. doi: 10.1007/978-3-540-74958-5_39. Series Title: Lecture Notes in Computer Science.

[53] Adrian Weller. Transparency: Motivations and Challenges. In Wojciech Samek, Grégoire Montavon, Andrea Vedaldi, Lars Kai Hansen, and Klaus-Robert Müller, editors, *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pages 23–40. Springer International Publishing, Cham, 2019. ISBN 978-3-030-28954-6. doi: 10.1007/978-3-030-28954-6_2.

[54] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[55] Yun Chi, Yi Xia, Yirong Yang, and R.R. Muntz. Mining closed and maximal frequent subtrees from databases of labeled rooted trees. *IEEE Transactions on Knowledge and Data Engineering*, 17(2):190–202, February 2005. ISSN 1041-4347. doi: 10.1109/TKDE.2005.30.

[56] Yun Chi, Yi Xia, Yirong Yang, and R.R. Muntz. Mining closed and maximal frequent subtrees from databases of labeled rooted trees. *IEEE Transactions on Knowledge and Data Engineering*, 17(2):190–202, February 2005. ISSN 1041-4347. doi: 10.1109/TKDE.2005.30.

[57] Mohammed J Zaki. Efficiently Mining Frequent Embedded Unordered Trees. *Fundamenta Informaticae*, page 20, 2005.

[58] Mohammed J. Zaki. TreeMiner: An Efficient Algorithm for Mining Embedded Ordered Frequent Trees. In *Advanced Methods for Knowledge Discovery from Complex Data*, pages 123–151. Springer London, London, 2005. ISBN 978-1-85233-989-0 978-1-84628-284-3. doi: 10.1007/1-84628-284-5_5.

[59] Zhi-Hua Zhou. *Ensemble Methods: Foundations and Algorithms.* CRC Press, June 2012. ISBN 978-1-4398-3003-1.